Scalable Virtual Data Infrastructure

Endbericht der Projektgruppe 553

Technische Universität Dortmund - Fakultät für Informatik

30. August 2012



Teilnehmer: André Bargull, Florian Bellmann, Tobias Friemel, Dmytro Galytskyy, Andriy Kandyba, Christoph Kröger, Long Phan, Stefan Plaga, Dimitrij Urin

Betreuer: Florian Feldhaus, Thomas Röblitz

Inhaltsverzeichnis

1. Einleitung und Motivation

In vielen Forschungsbereichen werden heute auf leistungsfähigen Rechnern Simulationen zu verschiedensten Zwecken durchgeführt. Das gilt sowohl für Ingenieurswissenschaften als auch für Naturwisschenschaften wie die Physik. Durch die Simulationen werden dann dort große Datenmengen erfasst, die es maschinell auszuwerten gilt. Diese Daten werden heute zumeist in Rechenzentren, die häufig Grid-Technologien zur Verwaltung der Rechen- und Speicherressourcen einsetzen, gelagert und verarbeitet. Zum Transfer und Verarbeitung der Daten setzen Wissenschaftler verschiedene Softwarewerkzeuge ein und nutzen zudem oft die Ressourcen anderer Forschungseinrichtungen. Wissenschaftler müssen daher nicht nur von ihrer aktuellen Forschung Kenntnis haben, sondern auch von den diversen Softwarewerkzeugen welche zur Datenanalyse notwendig sind. Hierzu zählen unter anderem GridFTP, GridTorrent und verschiedene Skriptsprachen.

Einige Forschungsbereiche wie die Hochenergiephysik oder die technische Chemie benötigen große Mengen an Speicher und Rechnenleistung, die selbst heute, wo die Hardware vergleichsweise günstig geworden ist, immer noch so hohe Kosten verursachen, dass sich nicht jedes Forschungsinstitut ein eigenes Rechenzentrum leisten kann. Im Zuge der internationalen Vernetzung ist es Instituten möglich auf Rechen- und Speicherressourcen zuzugreifen, die mehrere hundert Kilometer vom Forschungsstandort entfernt sind, um diese Ressourcen für ihre Arbeit zu verwenden. Allerdings haben die globale Vernetzung und die verschiedenen Softwareprodukte zur Verwaltung der Rechenzentren den Nachteil, dass Wissenschaftler und ihre Mitarbeiter ein Sammelsurium von Softwarewerkzeugen kennen und verwalten müssen. Diese Tatsache behindert die Forschung und kostet Zeit.

Unsere Projektgruppe sollte hierfür ein Werkzeug erstellen, mit dessen Hilfe es Wissenschaftlern möglich ist mit verschiedenen Speicherumgebungen wie lokale Speicher, Grid-Umgebungen oder auch Cloud-Umgebungen zu kommunizieren. Die Software soll es Wissenschaftlern ermöglichen Daten zu lesen und zu verändern, mit anderen Wissenschaftlern zu teilen und Daten mit weiteren Informationen zu versehen, die anderen Forschern Aufschluss über die Güte oder den Verarbeitungsgrad der Daten geben. Es soll eine Software geschaffen werden, die es erlaubt komfortabel mit Daten aus unterschiedlichen Rechenzentren zu arbeiten.

1.1. Anwendungsfälle

Mittels Anwendungsfällen sollen typische Arbeitsabläufe in der späteren Betriebsumgebung eines Softwareproduktes abgebildet werden. Dies ermöglicht es den Softwareentwicklern einen frühen Einblick in den späteren Verwendungszweck einer Software zu gewähren, so dass dieser für alle am Entwicklungsprozess beteiligten Personen erkenntlicher wird. Hierfür werden bereits während der Erstellung der Szenarien die später von der Software zu erfüllende Anforderungen formuliert. Nach Fertigstellung der Software kann dann anhand der zuvor formulierten Szenarien geprüft werden, inwieweit diese Anforderungen erfüllt wurden, oder ob es noch Abweichungen gibt, die es zu beheben gilt. Im Folgenden werden nun die Szenarien beschrieben, die im Rahmen der Projektgruppe erarbeitet wurden.

Szenario 1: *Ein Wissenschaftler möchte von zuhause aus mit seinem privaten Rechner, auf dem ein geeignetes Zertifikat installiert ist, auf Grid-Ressourcen seiner Universität zugreifen. Dabei möchte er Dateien von dem privaten Rechner in das Grid hochladen, dort eine Anwendung darauf ausführen und die Ergebnisse anschließend auf seinen Privatrechner herunterladen, um diese dort zu begutachten.*

Anforderungen:

- Einfacher Datentransfer von einem lokalen Rechner zu einer entfernten Ressource und umgekehrt
- Non-simultaner Zugriff einer Person auf mehrere Daten

Szenario 2: *Ein Wissenschaftler hat Messdaten aus einem realen Experiment gesammelt. Er möchte diese nun anderen Wissenschaftlern seines Fachgebiets zur Verfügung stellen. Diese sollen die Daten jedoch nicht verändern dürfen. Schreibender Zugriff auf die Daten ist daher lediglich ihm sowie ggf. anderen Mitgliedern seines Teams vorbehalten.*

Anforderungen:

- Einfaches, gruppenbasiertes Rechtemanagement
- Gleichzeitiger Zugriff mehrerer Personen auf die gleichen Daten
- Synchronisierung von Änderungen

Szenario 3: *Ein Wissenschaftler hat eine Vielzahl von Daten auf verschiedenen verteil-*

ten Ressourcen liegen. Er möchte nun auf eine Datei zugreifen, weiß jedoch nicht, auf welcher physikalischen Ressource sie sich genau befindet und möchte sich nun nicht auf jede Ressource einzeln begeben müssen, um dort eine Suche von Hand nach der Datei auszuführen.

Anforderungen:

- Gleichzeitiger Zugriff auf mehrere entfernte Ressourcen
- Suchfunktion für Daten

Szenario 4: *Ein Wissenschaftler führt eine Simulation auf einer Vielzahl unterschiedlicher Datensätze durch. Er möchte nun die Daten markieren, die er bereits zu Simulationszwecken genutzt hat. Zusätzlich möchte er die jeweils erzielten Ergebnisse in einigen Stichpunkten beschreiben. Bei der Auswahl der Daten für die nächste Simulation möchte er nun ausschließlich Datensätze angezeigt bekommen, die er zuvor noch nicht genutzt hat. Die Berechnung der Simulation erfolgt dabei in einem bestimmten Rechenzentrum. Die Daten selbst liegen jedoch auf Ressourcen in der ganzen Welt, auf die er zwar lesenden, jedoch keinen schreibenden Zugriff hat. Zudem geschieht der Dateizugriff über unterschiedliche Protokolle. Der Wissenschaftler arbeitet dabei auch von seinem Privatrechner aus, der über keine schnelle Internetverbindung verfügt. Daher muss der Wissenschaftler in der Lage sein die Daten ohne Umwege von ihren jeweiligen Ressourcen ins Rechenzentrum zu kopieren.*

Anforderungen:

- Eigene Metadaten hinzufügen und in Suchfunktion verwenden
- Komplexer Datentransfer von Ressourcen zu Ressourcen

Szenario 5: *Ein Wissenschaftler befindet sich häufig an unterschiedlichen Orten, von denen aus er auf verschiedene verteilte Ressourcen zugreift. Der Zugriff erfolgt dabei sowohl über berufliche als auch private Rechner. Er möchte jedoch, dass alle Änderungen, die er von einem Endgerät aus vorgenommen hat, auch dann noch verfügbar sind, wenn er sich später von einem anderen Gerät aus einwählt.*

Anforderungen:

- Verfügbarkeit und Synchronisation von Metadaten, Einstellungen, Präferenzen, etc. des Clientprogramms auf verschiedenen Rechnern

1.2. Anforderungen

> *Requirements-Engineering ist die Disziplin zur systematischen Entwicklung einer vollständigen, konsistenten, und eindeutigen Spezifikation, in der beschrieben wird, was ein softwaregestütztes Produkt tun soll (aber nicht wie), und die als Grundlage für Vereinbarungen zwischen allen Betroffenen dienen kann. Diese Disziplin umfasst Methoden, Beschreibungsmittel und Werkzeuge zur Ermittlung, Formulierung und Analyse von Aufgabenstellungen und Anforderungen an Systeme.*[Par98]

Für das Requirements-Engineering in der SVDI wurde eine Anforderungsspezifikation in Form eines Pflichtenhefts erstellt. Hierzu wurden die Anforderungen aus den bereits vorher erstellten Szenarien generiert und in Muss- und Kann-Kriterien unterteilt. Muss-Kriterien legen hier die zwingend erforderlichen Eigenschaften der Software fest, ohne die die Software nicht in ihrem späteren Arbeitsumfeld sinnvoll eingesetzt werden kann. Die Nichterfüllung eines einzelnen Muss-Kriteriums ist dadurch zumeist als Nichterfüllung der gesamten Entwicklung anzusehen. Kann-Kriterien hingegen stellen zusätzliche Eigenschaften dar, die die Software optionalerweise ebenfalls erfüllen sollte. Falls ein oder mehrere Kann-Kriterien nicht erfüllt werden, führt dies nicht automatisch zur Nichterfüllung der Gesamtentwicklung. Darüber hinaus wurden auch Abgrenzungskriterien definiert, welche aufzeigen sollen, was für Eigenschaften die Software nicht erfüllen soll bzw. muss. Das Pflichtenheft wurde anschließend den Betreuern vorgelegt, so dass diese prüfen konnten, ob alle Anforderungen entsprechend beachtet wurden. Im Folgenden sind die Kriterien aus dem Pflichtenheft aufgelistet.

1.2.1. Musskriterien

Virtual Storage Layer (VSL)

M1 Es muss ein Virtual Storage Layer implementiert werden. Dieser soll den Zugriff auf die entfernten Daten vereinheitlichen und den Komfort der Softwarenutzung erhöhen.

M2 Es muss ein Schnittstellendokument erstellt werden, welches wie ein Nachschlagewerk verwendet werden kann, um beispielsweise erweiternde Softwareprodukte zu schaffen, die den VSL erweitern oder anderweitig nutzen.

Daten

M3 Die SVDI muss die Datenverwaltung jedes einzelnen Benutzers unterstützen. Es

muss gewährleistet werden, dass die Daten eines Benutzers auf dem entfernten Datenträger von eben diesem Benutzer gelesen und geändert werden können.

M4 Es muss eine Suchfunktion existieren, die es dem Benutzer ermöglicht die sichtbare Datenmenge zu filtern, um die von ihm gewünschten Daten schnell zu finden.

M5 Es muss für die Anwender möglich sein aus einer Menge von Dateien einzelne lesen zu können. Beispielsweise gehören mehrere hundert Dateien zu einer insgesamt 1 GB großen Auswertung. Es muss dem Wissenschaftler nun möglich sein einige Dateien auswählen und laden zu können, ohne dass die komplette Auswertung transferiert wird.

M6 Es muss dem Anwender eine Möglichkeit geboten werden die Daten mit Metadaten zu versehen.

Metadaten

M7 Es muss ein Metadatendienst existieren, der die zentrale Verwaltung der Metadaten übernimmt und für deren Verfügbarkeit Sorge trägt.

M8 Es muss möglich sein einfache Operationen wie das Anlegen, Löschen und Ändern auf den Metadaten auszuführen.

M9 Es muss ein Mindestmaß an Redundanz der Metadaten gewährleistet werden.

M10 Dem Anwender muss eine Möglichkeit gegeben werden auf den Metadaten zu suchen, und die Ergebnisliste einschränken zu können.

User-Interface

M11 Es muss ein Interface implementiert werden über das es den Benutzern möglich ist auf ihre Daten zuzugreifen, Datenquellen einzusehen, Informationen über Ressourcen zu beziehen, Zugriffe zu kontrollieren, Änderungen durchzuführen und Daten aus verschiedenen Speicherquellen zu beziehen.

Zugriffsrechte

M12 Es muss gewährleistet sein, dass ein Benutzer die Zugriffsrechte auf die Daten von denen er Autor ist, ändern kann. Mittels eines Authentifizierungssystems muss es möglich sein die Identität des Benutzers festzustellen. Darüber hinaus muss der Zugriff eines Benutzers auf seine und ihm freigegebene Daten autorisiert werden können.

Weitere Anforderungen

M13 Es muss eine Unterstützung von Basisszenarien ermöglicht werden. Im Rahmen des Projektes werden typische Arbeitsabläufe definiert, die die Software unterstützen muss.

M14 Die Durchführung von Arbeitsabläufen muss für den Benutzer verständlich und logisch sein, so dass bisher komplizierte Abläufe, in die mehrere Systeme involviert sind, vereinfacht werden.

M15 Es müssen unterschiedliche Technologien für die Datenaufbewahrung unterstützt werden. Mittels der SVDI müssen verschiedene Speicherorte, wie sie derzeit verwendet werden um Daten zu persistieren, nutzbar sein.

M16 Zum Projektabschluss muss eine zum Projekt passende Dokumentation vorliegen.

M17 Die Software soll plattformunabhängig entworfen werden, muss aber nur auf mindestens einem gängigen Betriebssystem lauffähig sein.

M18 Mittels der zu entwickelnden Software muss es möglich sein Ressourcen an der TU Dortmund nutzen zu können.

1.2.2. Kannkriterien

Folgende Anforderungen sollen für die zu entwickelnde Software optional sein:

K1 Es könnte der gleichzeitige Zugriff auf beliebig viele verschiedene Grids und Clouds ermöglicht werden. Damit ist die Anbindung an eine größere Anzahl verschiedener entfernter (Datei-)systeme, als der in den Muss-Kriterien beschriebenen, gemeint.

K2 Es könnte eine Einbindung des Systems in das lokale Dateisystem ermöglicht werden, um die Ressourcen über Betriebssystem eigene Mittel transparent verwalten und verwenden zu können.

K3 Es könnte eine Schnittstelle zur Konfiguration für externe Programme durch eine API bereitgestellt werden.

K4 Es könnte ein plattformunabhängiger Client für das System entwickelt werden.

K5 Es könnte dem Benutzer eine Möglichkeit im System bereitgestellt werden, um Kopien für Backups seiner Daten/Metadaten zu erstellen.

K6 Es könnte dem Benutzer ermöglicht werden seine persönlichen Einstellungen zentral und nicht nur lokal zu sichern.

K7 Es könnte eine Anbindung für das System entwickelt werden, um die Verwendung verbreiteter kommerzieller Dienste (z.B. Dropbox, Amazon S3, ...) zu ermöglichen.

K8 Es könnte eine Verschlüsselung für die Daten und Metadaten implementiert werden, um die Sicherheit bei der Datenhaltung und beim Datentransfer zu gewährleisten.

K9 Es könnten Caching-Methoden verwendet werden, um das Arbeiten auf entfernten Daten performanter zu gestalten.

K10 Es könnte ein spezielles Verhalten für Verbindungsabbrüche definiert werden, um Datenverlust zu vermeiden.

K11 Es könnten Anpassungen für die verschiedenen Fachbereiche in die Benutzeroberfläche einfließen, um die Ergonomie für die Anwender zu erhöhen. Dazu sollten Fachtermina im Programm verwendet werden.

K12 Es könnte eine Internationalisierung für die Benutzeroberfläche implementiert werden.

K13 Es könnte eine umfangreichere Rechteverwaltung für Daten und Metadaten als in den Muss-Kriterien umgesetzt werden.

1.2.3. Abgrenzungskriterien

Folgende Anforderungen sollen für die zu entwickelnde Software ausgeschlossen sein:

A1 Es soll keine Verwaltung von Rechenkapazitäten (im Grid, Cloud, etc.) erstellt werden.

A2 Es soll keine Jobzuweisung auf einzelne Rechner im Grid, der Cloud, usw. durch die Software erfolgen.

A3 Es soll keine Software für vollautomatische, redundante Speicherung oder andere Backup- und Synchronisierungsmechanismen gebaut werden.

A4 Es soll keine Neuimplementierung eines verteiltes Dateisystems erstellt werden.

A5 Es soll keine spezielle Anpassung an einen einzelnen Nutzerkreis erfolgen.

A6 Es soll nicht zu einer Ausrichtung an Consumerservices kommen (vgl. vorher. Punkt).

A7 Es soll keine eigene Grid-ähnliche Architektur konzipiert und aufgebaut werden.

2. Organisation und Arbeitsweise

Im Folgenden soll beschrieben werden wie sich die Projektgruppe über die Zeit organisiert hat, und in welcher Art und Weise sich die gewählte Organisationsform in der Arbeitsweise niedergeschlagen hat. Dazu sollen nun die drei Hauptbereiche „Treffen“, „Projektplanung“ und „Kommunikation“ genauer betrachtet werden.

2.1. Treffen

Neben den wöchentlichen Treffen der Projektgruppe fanden zudem Vorbereitungstreffen und eine Kick-Off Phase statt. Diese drei sollen nun genauer erläutert werden.

2.1.1. Vorbereitungstreffen und Kick-Off Phase

Vor Beginn des ersten Projektgruppensemesters gab es zwei Vorbereitungstreffen. Im ersten Treffen, das Ende Januar 2011 statt fand, haben sich die Projektgruppenteilnehmer und die Betreuer zum ersten Mal persönlich getroffen. Neben dem Aspekt des Kennenlernens wurde im Treffen die Vorbereitung zum zweiten Treffen behandelt, sowie die grundsätzlichen Anforderungen hinsichtlich der späteren Organisation der Projektgruppe seitens der Betreuer besprochen.

Nach diesem Treffen startete eine mehrwöchige Seminarphase, in welcher jeder Teilnehmer eines oder mehrere Themen für das zweite Vorbereitungstreffen vorzubereiten hatte. Die Seminarphase endete mit dem zweiten Treffen, welches als Kick-Off Treffen für die eigentliche Projektgruppenarbeit vorgesehen war. Neben der Besprechung der Seminarthemen zur Vertiefung in den Aufgabenbereich des Projektgruppenthemas, wurde im Kick-Off Treffen die weitere Organisationsform der Projektgruppe erörtert und schlussendlich auch die Projektleitung von den Teilnehmern gewählt. Wie auch im ersten Treffen oblag die Leitung des Treffens noch den Betreuern.

2.1.2. Wöchentliche Meetings

Während der Vorlesungszeit, sowie auch während der vorlesungsfreien Zeit zwischen dem Wintersemester 2011/12 und dem Sommersemester 2012, fanden wöchentliche Projektgruppentreffen mit allen Teilnehmern zusammen mit den Betreuern statt. Hierbei wurden im ersten Projektgruppensemester noch zweimal wöchentlich Treffen abgehalten, später hingegen nur noch eimal pro Woche. Grund für die Änderung auf ein Treffen pro Woche war die stärkere Fokussierung auf die eigentliche Entwicklung der Anwendung, wo hingegen im ersten Semester der Fokus eher auf der Konzeption der Architektur o.ä. Aufgabenbereiche lag.

Ein weiterer Grund im ersten Semester zwei Treffen pro Woche abzuhalten, war eine wie folgt vorgesehene Themententrennung. Das erste Treffen innerhalb der Woche sollte für allgemeine Aufgaben wie Planung und Organisation genutzt werden, während im zweiten Treffen vor allem technische Gesichtspunkte besprochen werden sollten. Diese angedachte Trennung hat sich für die Projektgruppe jedoch nicht bewährt, so dass sie letztendlich aufgegeben wurde und beide Treffen für alle Themenbereiche genutzt werden konnten.

Die Leitung der wöchentlichen Treffen übernahmen nicht mehr, wie noch bei den Vorbereitungstreffen, die Betreuer, sondern die Projektgruppenteilnehmer selbst. Hierzu wurde ebenso entschieden, dass die Projektleitung sowohl für den Vorsitz, als auch für die Protokollierung der Treffen zuständig sei. Die Rolle der Betreuer während der Meetings bestand daher vor allem in beratender Position.

Inhaltlich befassten sich die zweistündigen Treffen mit einem breiten Spektrum von Themen. Zum einem wurden sie natürlich dazu genutzt die Gesamtplanung des Projekts untereinander abzusprechen, d.h. zu organisieren wer welche Aufgaben zu erledigen hat, wie weit der Fortschritt in den einzelnen Aufgaben ist oder über mögliche Probleme bei der Aufgabenbearbeitung zu sprechen und im Team dann entsprechend zu versuchen diese zu lösen. Zum anderen wurden in den Meetings auch immer wieder Vorträge über Technologien oder Tools gehalten, welche interessant oder nützlich für die weitere Projektbearbeitung sein könnten.

2.2. Projektplanung

Die Aufgabenkoordination und -planung wurde von der Projektleitung durchgeführt. Diese wurde von den Teilnehmern im Anschluss an das Kick-Off Treffen selbst gewählt (vgl. o.). Die Projektleitung selbst bestand im ersten Semester aus zwei Projektgruppenteilnehmern. Im zweiten Semester wurde die Leitung auf einen einzigen Projektleiter im Zuge einer weiteren Aufgabenumstrukturierung umverteilt.

Die Vergabe der weiteren Rollen bzw. Aufgaben an die restlichen Teilnehmer erfolgte im ersten Semester nach verschiedenen Kriterien. Insbesondere waren die einzelnen Aufgaben verschieden stark gewichtet, so dass sich einzelne Teilnehmer z.B. um die Recherche von benötigten Softwarekomponenten kümmerten, während andere für die Konzeption der Gesamtarchitektur Sorge zu tragen hatten. Eine stringentere Einteilung in schärfer umrissene Arbeitsbereiche fand erst im zweiten Semester statt. Dazu wurden die folgenden vier Verantwortlichkeitsbereiche geschaffen:

- Implementierung Server
- Implementierung Client
- Test + Integration
- Dokumentation

Da diese Verantwortlichkeitsbereiche entscheidend für den weiteren Verlauf der Projektgruppe waren, sollen sie nun genauer erklärt werden.

Jeder Teilnehmer, mit Ausnahme der Projektleitung, wurde zu einem der Bereiche zugeteilt. Hierbei stand jeder Arbeitsgruppe zudem ein Gruppenleiter vor. Dieser stimmte die Aufgaben innerhalb der Gruppe genauer ab, so dass die Projektleitung entlastet werden konnte und sich besser um die Gesamtkoordinierung kümmern konnte. Zudem oblag es den Gruppenleitern für ihren jeweiligen Bereich Arbeitspläne zur längerfristigen Projektplanung zu gestalten. Anhand dieser wurden u.a. auch entsprechende Milestones definiert. Zur Überwachung des Arbeitsfortschritts waren die Teilnehmer weiterhin angewiesen in wöchentlichen Rundmails an die Gruppe ihre Arbeit zu dokumentieren, um ggf. frühzeitig bei möglichen Problemen reagieren zu können. Die Gruppenleiter wiederum waren verpflichtet den Gesamtfortschritt innerhalb ihrer Gruppe an die Projektleitung weiterzugeben.

Der Aufgabenbereich „Implementierung Server“ befasste sich mit der Realisierung der Serverkomponente der Anwendung. Dies umfasste sowohl die eigentliche Programmierung des Servercodes, aber auch die Konzeption der Serverschnittstellen und ebenfalls die Installation und Konfiguration der Datenbanken und des Servers.

Im Bereich „Implementierung Client“ wurde entsprechend ihres Namens die Implementierung der SVDI-Client Komponente umgesetzt. Wie auch beim Server bestand die Arbeit nicht nur aus der Programmierung. Die Arbeitsgruppe musste zusätzlich die Oberfläche (GUI) des Clients konzipieren, eine Validierung gegen die vom Server angebotenen Schnittstellen durchführen, die Funktionsweise des Clients in dem Benutzerhandbuch dokumentieren usw.

Die beiden anderen Verantwortlichkeitsbereiche „Test + Integration“ und „Dokumenta-

tion" umfassten die restlichen Tätigkeiten der Projektgruppe. Dies war zum einen die Erstellung und Durchführung von Tests sowohl des Clients als auch des Servers. Dazu wurde gemäß der festgelegten Schnittstellen zwischen Server und Client die Implementierung auf Funktionsfähigkeit geprüft. Zum anderen befasste sich die „Integration" mit der Zusammenführung beider SVDI-Komponenten und auch dem Aufbau der initialen Entwicklungsumgebung. Schließlich wurde im Bereich „Dokumentation" die gesamte Zusammenführung aller Dokumentationsfragmente aus allen Aufgabenbereichen koordiniert, um aus diesen den Endbericht sowie die weiteren begleitenden Dokumentationsartefakte zu erstellen.

2.3. Kommunikation

Neben den bereits beschriebenen wöchentlichen Treffen wurden zur Kommunikation zwischen den Projektgruppenteilnehmern mehrere verschiedene Kommunikationskanäle installiert.

2.3.1. E-Mails und Jabber

Zum Anfang der Projektgruppe wurde eine Mailingliste erstellt über die die Teilnehmer als auch die Betreuer die gesamte Projektgruppe erreichen konnten. Neben E-Mails für die allgemeine Koordination wurden über die E-Mailliste auch die im Abschnitt „Projektplanung" erwähnten Rundmails gesendet. D.h. über die Liste wurden von den Gruppenmitgliedern bzw. später den Gruppenleitern die Wochenpläne verschickt, in denen die jeweiligen geplanten Arbeitsaufgaben für die Gesamtkoordinierung dargestellt wurden. Daneben wurden am Wochenende die tatsächlich erfüllten Arbeitsaufgaben ebenfalls über die Mailingliste kommuniziert.

Als zusätzliche Kommunikationsmöglichkeit wurde ein Jabber-Server auf einem Server der TU Dortmund aufgesetzt. Damit stand der Projektgruppe auch ein direkterer Kommunikationskanal im Vergleich zu der Mailingliste zur Verfügung. Jabber wurde vor allem dann genutzt, wenn eine schnelle Koordination zwischen wenigen Teilnehmern nötig war. Wenn hingegen die gesamte Gruppe informiert werden sollte, wurde weiterhin auf E-Mails bzw. die anderen Kommunikationsmittel zurück gegriffen.

2.3.2. Redmine

Ein weiterer Kommunikationskanal war das Projektmanagement-Tool „Redmine". Redmine wird vom ITMC der TU Dortmund bereitgestellt und steht dort für Projektarbeiten im

Rahmen von Forschung und Lehre zur Verfügung. Redmine stellt hierbei mehrere Dienste bereit: ein Wiki, eine Dokumentenablage, ein Ticketsystem, und auch ein Source-Code Repository zur Versionsverwaltung über SVN.

Das Wiki und die Dokumentenablage wurde von der Projektgruppe zumeist für Information genutzt, welche längerfristig zur Verfügung stehen mussten, oder wenn eine Versionshistorie benötigt wurde. Beispiele hierfür sind die Einteilung der Teilnehmer in die Aufgabenbereiche oder auch das Bereitstellen von Installations- und Gebrauchsanweisungen. Genutzt wurde das Wiki damit vor allem im ersten Semester, als noch die grundlegende Informationsrecherche im Vordergrund stand.

Das Ticketsystem von Redmine wurde hingegen vor allem im zweiten Semester, im Rahmen der eigentlichen Entwicklung, genutzt, um dort eine bessere Abstimmung zwischen den einzelnen Arbeitsgruppen zu ermöglichen. So konnten dann für zu erledigende Aufgaben immer entsprechende Tickets angelegt und mittels dieser der Arbeitsfortschritt überwacht werden. Erstellt wurden die Tickets entweder von einem einzelnen Gruppenmitglied, wenn er für sich selber oder für jemand anderen eine Aufgabe hatte, oder sie wurden im Rahmen der wöchentlichen Meetings angelegt, wenn dort die Aufgabenplanung für die Woche stattfand. Im wöchentlichen Meeting wurde zudem der Fortschritt der noch offenen Tickets besprochen und bereits erledigte Tickets wurden geschlossen. Neben den üblichen Attributen für Ticketsysteme wie Titel, Beschreibung, Bearbeiter und Status bietet Redmine auch eine Time-Tracking Funktionalität an, so dass für jedes Ticket beschrieben werden kann, wann das jeweilige Beginn- und Abgabedatum ist und wie groß der geschätzte Aufwand ist bzw. wie viel Zeit effektiv aufgewendet wurde. Während das Abgabedatum vor allem für die Planung des Projekt- bzw. Gruppenleiters genutzt wurde, konnten die restlichen Teilnehmer über den Vergleich von geschätztem Aufwand gegenüber dem effektiv aufgewendeten Aufwand in erster Linie für sich selbst überprüfen, wie weit sie mit ihrer Aufwandsschätzung richtig lagen.

Für die Versionsverwaltung des Source-Codes wurde SVN genutzt. Hierzu wurde vom Integrationsmanager eine initiale Projektstruktur und Verzeichnishierarchie angelegt. Für die anschließende Benutzung von SVN sind dann entsprechende Commit-Richtlinien verfasst und im Wiki abgelegt worden. Unter anderem wurde in diesen festgelegt, dass einzelne Commits immer eine Beschreibung der Änderungen enthalten müssen, so dass für die restlichen Projektgruppenteilnehmer offensichtlich gemacht wird, warum genau diese Änderung getätigt wurde. Es wurde ebenso festgelegt, dass in diesen Commit-Beschreibungen die Tickets referenziert werden, zu denen die Änderung gehört, sofern solche Tickets vorhanden waren.

2.4. Schlussfolgerungen

Zusammengefasst wurde die vorgenommene Organisation und Arbeitsweise von den Projektgruppenteilnehmern überwiegend als positiv bewertet. Die anfänglich vorgenommene Aufteilung der Projektleitung in zwei Personen half stark dabei mit die Gesamtorganisation geregelter aufzubauen als dies vielleicht mit nur einer Person möglich wäre. Insbesondere da noch keinerlei Projektleitungserfahrung in der Gruppe vorlag. Auch wurden die wöchentlichen Treffen entscheidend für den Fortschritt in der Projektgruppe angesehen, da sich dort Probleme und Fragen schneller und direkter lösen lassen konnten im Vergleich zu einer rein elektronischen Kommunikation wie E-Mails. Die Arbeitseinteilung und -dokumentation über Ticketmanagementsysteme war für einige Teilnehmer zu Beginn der Projektgruppe noch unbekannt. Rückblickend wäre hier eine bessere Einführung in diese Arbeitsweise von Vorteil gewesen, mit dem Ziel bereits früher und noch stärker mittels Tickets die Arbeit einzuplanen. Auch die restlichen Planungs- und Organisationsmittel, welche erst im zweiten Semester eingesetzt worden sind, wie zum Beispiel die Erstellung von Entwicklungs- und Wochenplänen, hätten bereits im ersten Semester eingesetzt werden sollen, da diese die Kommunikation zwischen den Projektgruppenteilnehmern viel stärker anregten.

3. Verteilte Systeme

Während der Einarbeitungsphase zur Projektgruppe wurden mehrere Vorträge zum Thema „Verteilte Systeme“ und anderer verwandter Technologien und Standards gehalten. Die Inhalte und Ergebnisse der Vorträge sollen hier nun kurz beschrieben werden.

3.1. Peer-to-Peer

3.1.1. Einführung

Peer-to-Peer-Connection heißt, dass direkte Verbindungen zwischen mehreren Rechnern hergestellt werden. In einem Peer-to-Peer-Netz werden alle Rechner als gleichberechtigt angesehen und können sowohl Dienste anbieten, als auch Dienste von den anderen Rechnern, auch Peers genannt, in Anspruch nehmen. Wenn sich die Peers untereinander in einem Netzwerk anhand ihrer Objekt-ID identifizieren können, spricht man von strukturierten Overlays. Unstrukturierte Overlays zeichnen sich dagegen dadurch aus, dass es keine Zuordnungsstruktur gibt und die Peers nur nach einem oder mehreren Kriterien gesucht werden können.
Es werden im Allgemeinen folgende Typen von Peer-to-Peer-Systemen unterschieden:

- *Zentralisierte Peer-to-Peer-Systeme*: Bei solchen Systemen wird ein Server für die Indexierung und Anmeldung der Peers benötigt. Die Peers müssen sich zuerst auf dem Server registrieren, um dann mit den anderen Peers im Netzwerk kommunizieren zu können.

- *Dezentralisierte Peer-to-Peer-Systeme*: Das gesamte Netzwerk organisiert sich selbstständig. Eine Möglichkeit dies zu realisieren geschieht über verteilte Hashtabellen. Dieser Ansatz soll nun genauer erläutert werden.

Distributed Hash Tables(DHT) gehören zur Klasse der dezentralisierten Peer-to-Peer-Systeme und bieten einen Lookup Service. Der Lookup Service ist wie eine Hashtabelle aufgebaut, in der Information über alle im Netzwerk beteiligten Peers gespeichert ist. Damit sollen DHT auch bei einer großen Anzahl von Knoten gut skalieren.

Zu den wichtigsten Operationen eines P2P-Netzes zählen beispielsweise das Einfügen, Entfernen und Auffinden von Objekten, sowie das Einfügen und Entfernen von Peers. Durch die Verwendung von Hashtabellen sind schnelle und effiziente verteilte Operationen im Netzwerk möglich.[p2p]

Vorteile:

+ Dezentralisierung: Knoten können problemlos ins Netz kommen.

+ Hoher Durchsatz beim Datentransfer: Die gleiche Information kann von mehreren Rechnern gleichzeitig bezogen werden.

+ Beschränkte Ausfallsicherheit: Wenn ein Rechner ausfällt, kann es immer noch andere Rechner mit den gleichen Daten im Netz geben.

Nachteile:

- DHT können unübersichtlich werden und dadurch schwer manuell zu verwalten.

- Proprietäres Protokoll.

- Komplexe Rechteverwaltung.

3.1.2. Projekte

In der Seminarphase wurden mehrere auf P2P-Technologien basierende Projekte betrachtet. Dies sind „P2P for E-Science“ von der Universität München und „GridTorrent“ von der Universität Athen.

P2P for E-Science

Dieses Projekt soll Wissenschaftlern unabhängig ihres Forschungsbereiches wie Astrophysik, Geographie usw. bei ihrer täglichen Arbeit helfen. Der in diesem Projekt entwickelte Ansatz *HiSbase* basiert auf der P2P-Technologie und soll ein dezentrales und skalierbares Datenmanagement bieten. Hierzu werden die zur Verfügung stehenden Ressourcen innerhalb der Forschungsgruppe ausgenutzt. Dies betrifft u.a. den Speicher, aber auch die Rechenkapazität. Als ein Problem während der Umsetzung von „P2P for E-Science“ erwies sich laut den Entwicklern die Verteilung der Daten innerhalb des Netzes, da die

Wissenschaftler für ihre Forschung zum Teil gleichzeitig auf die gleichen Daten zugegriffen haben. Aus diesem Grund wurde zusätzlich zur DHT eine spezifische Verteilungsfunktion entwickelt.
Laut den Entwicklern ist das Projekt erfolgreich beendet worden [SGM+08].

GridTorrent

Das Ziel dieses Projektes war die Entwicklung eines effizienten, skalierbaren und robusten Datenmanagementsystems für Grid-Umgebungen. Zunächst wurden hierzu die Nachteile der üblichen Methoden zum Datentransfer in Grids wie GridFTP und Replica Services analysiert, um ebendiese Nachteile bei „GridTorrent“ zu vermeiden. Wichtig war es den Entwicklern hierbei eine dezentrale Struktur zu nutzen, womit sich wiederum Peer-to-Peer-Netzwerk und auch Distributed Hash Tables anboten.
Die Entwickler haben sich in diesem Projekt dann dazu entschieden ein neues, auf Torrent basierendes Protokoll zu entwickeln, das sie GridTorrent nannten. Um die Engpässe beim Replica Service zu vermeiden, wurde zudem eine Distributed Hash Table eingesetzt. Somit konnten laut den Entwicklern die Vorteile der Dezentralisierung ausgenutzt werden.[gri]

3.2. Grid-Computing

Grid-Computing beschäftigt sich mit der Speicherung und Verarbeitung von großen und verteilt liegenden Datenmengen. In diesem Abschnitt soll ein kurzer Überblick über die verschiedenen Arten von Grids, Architektur und Sicherheit, sowie häufig genutzte Grid-Middlewares gegeben werden. Grid Computing hat in der Regel die folgenden Eigenschaften:

- **Interoperabilität** - Ein Grid integriert und koordiniert dezentrale Ressourcen aus unterschiedlichen, kontrollierten Domains und kommuniziert in der Regel über das Internet.

- **Gemeinsame Ressourcennutzung** - Die wichtigsten Ressourcen im Grid sind Rechenleistung, Daten und die Netzwerkinfrastruktur. Jedoch werden zwischen Grid-Teilnehmern auch andere Dinge, wie Sensoren, wissenschaftliche Instrumente sowie Arbeitszeit der Wissenschaftler und Administratoren geteilt, um dadurch verschiedenste Aufgaben zu lösen.

- **Virtuelle Organisation** - Bei Virtuellen Organisationen (VOs) handelt es sich um Gruppen von Personen oder Instituten, die ähnliche Ziele verfolgen und sich bestimmten Regeln bezüglich der Nutzung der Grid-Infrastruktur unterwerfen.

- **Skalierbarkeit** - Grids bündeln die Rechenleistung vieler einzelner Rechner und bilden so einen virtuellen „Supercomputer“. Standardisierte Protokolle und Schnittstellen ermöglichen es, dem Grid problemlos neue Ressourcen hinzuzufügen und dessen Leistungsfähigkeit zu verbessern.

Grids werden in der Regel von mehreren Personen oder Forschungsgemeinschaften betrieben. Der Resource-Provider stellt so z.B. die Hardware und Netzwerkanbindung zur Verfügung, während der Grid-Provider sich um die Administration des Grids kümmert. So ist der Grid-Provider auch dafür verantwortlich, den Nutzern des Grids den Zugriff auf dessen Ressourcen zu gewähren und Rechenzeit sowie Speicherplatz zuzuteilen. Der Zugriff auf Grids erfolgt in der Regel mittels geeigneter Middlewares, wie Globus oder gLite. Diese bieten zum einen die Möglichkeit, im Grid zu navigieren, Dateien zu manipulieren und Jobs auszuführen, zum anderen bieten sie auch Schnittstellen zur Batch-Programmierung. Da Grids von Wissenschaftlern, also der Zielgruppe der SVDI-Applikation, stark genutzt werden und über standardisierte Schnittstellen nach außen verfügen, waren diese von Beginn an eine interessante Zielplattform, die letztlich auch von der SVDI unterstützt wird.[IF01],[Fos02]

3.2.1. Grid-Arten

Grids lassen sich in verschiedene Kategorien einteilen, abhängig davon zu welchem Zweck die jeweiligen Ressourcen genutzt werden. Doch auch Mischformen sind weit verbreitet, so dass die Einteilung in der Regel nicht eindeutig ist.

- **Computational Grid** - Diese Gridform bündelt die Rechenleistung vieler CPUs, um parallel Probleme zu lösen.

- **Data Grid** - Diese Gridform ermöglicht dem Anwender die Speicherung von großen Datenmengen sowie den Zugriff auf verteilte Datenbanken.

- **Resource Grid** - Computational Grid und Data Grid werden zusammen als Resource Grid bezeichnet.

- **Weitere Typen** - Darüber hinaus gibt es auch noch andere Gridtypen, die weitere, oft sehr spezielle Dienstleistungen zur Verfügung stellen.

[Man06]

3.2.2. Sicherheit im Grid

Da Grids ihre Kommunikation meist über das Internet abwickeln, wobei hiermit sowohl die Kommunikation mit den Endbenutzern, als auch mit den einzelnen Rechnern im Grid selbst gemeint ist, ist Sicherheit für Grids von großer Bedeutung. In den Middleware-Komponenten (Globus, gLite, ...) kommt so z.B. GSI (Grid Security Infrastructure) zum Einsatz, welches auf einer X.509-Public-Key-Infrastructure aufsetzt. Um auf Grid-Ressourcen zugreifen zu können, ist es daher für jeden Nutzer erforderlich über ein gültiges Zertifikat einer vertrauenswürdigen Institution zu verfügen. Auf Basis dieses Zertifikats können Nutzer dann Proxy-Zertifikate generieren, welche über eine kürzere Laufzeit (oft nur wenige Stunden bis Tage) verfügen, sich ansonsten aber nicht von den eigentlichen Zertifikaten unterscheiden. Somit gehen der Nutzer und auch der Gridbetreiber ein geringeres Sicherheitsrisiko ein, als wenn sie die ursprünglichen, langlebigen Zertifikate über das Internet versenden, da ein solches Proxy-Zertifikat nur in seltenen Fällen mittels Brute-Force „geknackt" werden kann, ohne dass es bereits abgelaufen ist. Zur sicheren Übertragung von Daten wird SSL/TLS (Secure Socket Layer/Transport Layer Security) genutzt, das auf TCP in der Transportschicht im OSI-Schichtenmodell aufsetzt. Außerdem wird mit der GSS-API eine standardisierte Programmierschnittstelle für Sicherheitssoftware geboten.[BF06]

3.2.3. Dienste im Grid

Die in einem Grid-System laufenden Dienste können darin unterschieden werden, ob sie für den Betrieb des Grids selber benötigt werden, oder ob sie für andere administrative oder benutzergesteuerte Prozesse genutzt werden. Einige wichtige Grid-Dienste sollen im Folgenden kurz erläutert werden: Scheduling-Dienste für die automatische Verarbeitung von Jobs sowie die Koordination der dazu benötigten Ressourcen zuständig. Monitoring-Dienste überwachen den Status der Grid-Ressourcen und der Grid-Execution-Management-Dienst ist als Schnittstelle für das Entgegennehmen und Überwachen von Jobs durch den User zuständig. Außerdem gibt es noch Datentransfer- und Replikationsdienste, die Daten innerhalb des Grids speichern und synchronisieren.[IF01],[Fos02]

3.3. Webtechnologien

3.3.1. Webservices

Webservices sind Anwendungen die über ein Netzwerk, wie dem Internet, angesprochen werden können. Sie sind auf einem Server installiert und dort dann über URIs (Unified Resource Identifier) von einem Client aus erreichbar. Für den Datentransport werden

meist die Internetprotokolle HTTP beziehungsweise, im Falle von verschlüsselten Verbindungen, HTTPS eingesetzt. Dies bietet sich deshalb an, da diese Protokolle in der Regel von Firewalls nicht blockiert werden.

Ein Webservice selbst besteht aus einer Schnittstellenbeschreibung und dem Dienst an sich. Für auf dem SOAP-Protokoll basierende Webservices wird die Schnittstellenbeschreibung im WSDL-Format (Web Services Description Language) geschrieben. Das WSDL-Dokument enthält sämtliche vom Dienst angebotenen Funktionen mit den dazugehörigen Parametern. Weiterhin erlaubt die Schnittstellenbeschreibung mittels WSDL, dass entsprechende Entwicklungstools automatisch den passenden Quellcode für die jeweilige Programmiersprache generieren können, der zum Aufruf des Webservices benötigt wird. Damit können dann die vom Webservice angebotenen Funktionen auf die gleiche Art und Weise im Programm aufgerufen werden, wie dies für normale lokale Funktionen der Fall ist, während im Hintergrund der passende Webservice Aufruf erstellt wird. [web]

3.3.2. XML

Die Abkürzung XML steht für „Extensible Markup Language“ und bezeichnet eine vom W3C Konsortium spezifizierte Sprache zur Darstellung von Daten in Textform. Durch die Wahl alle Daten in Textform zu repräsentieren und damit auf Binärdaten zu verzichten, ermöglicht XML den einfachen Datenaustausch in auch heterogenen Systemen und erlaubt es zudem, dass die Daten in einer auch für Menschen lesbaren Form vorliegen. [xml]

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <dokument>
3   <einträge>
4     <eintrag id="1">Inhalt</element>
5     <eintrag id="2"/>
6   </einträge>
7 </dokument>
```

Listing 3.1: Ein einfaches XML-Dokument

Anhand des Beispiel XML-Dokuments in 3.1 soll nun kurz die allgemeine Syntax von XML erklärt werden. Die XML-Spezifikation definiert drei Hauptstrukturelemente: Elemente, Attribute und Text. Elemente geben hierbei die allgemeine Struktur des XML-Dokumentes vor. Im Beispiel sind dies `<dokument>`, `<einträge>` und `<eintrag>`. Zusätzlich können Elemente mit Attributen versehen werden, wie im Beispiel das Attribut `id`. Weiterhin lassen sich Elemente verschachteln und es kann freie Textblöcke geben. Jedes XML-Dokument hat zudem genau ein Wurzelelement und ggf. eine XML-Deklaration (vgl. erste Zeile) in der u.a. die XML-Version und die Zeichenkodierung angegeben werden kann.

Über XML-Schemata kann der Aufbau eines XML-Dokuments definiert werden, d.h. es kann definiert werden welches Element an welcher Stelle stehen darf und was die erlaubten untergeordneten Elemente und Attribute sind. Insbesondere für den Datenaustausch wird oft ein Schema benötigt, so dass mit diesem überprüft werden kann, ob ein XML-Dokument *wohlgeformt* ist, daher ob es den festgelegten Regeln entsprechend des Schemas folgt.

3.3.3. EJB

Die Abkürzung *EJB* steht für Enterprise Java Beans und beschreibt eine Komponentenarchitektur, welche für die Implementierung der Geschäftslogik innerhalb einer Java Enterprise Umgebung verwendet werden kann. Dadurch dass mit EJB die Geschäftslogik vom Rest des Systems abgekapselt werden kann, ist es möglich komplexe und verteilte Softwaresysteme deutlich einfacher zu entwickeln. Durch die Bean Komponenten kann die -meist geheime- Geschäftslogik auch in Kombination mit Webservices eingesetzt werden ohne einen Informationsverlust befürchten zu müssen.

Eine Java Bean kann entweder als *local* oder als *remote* Bean gekennzeichnet werden. Eine lokale Bean kann nur von demselben System aufgerufen werden im dem sie erstellt wurde und ist daher nicht für externe Zugriffe freigegeben. Remote Beans können hingegen von entfernten Systemen angesprochen werden und können damit zum Beispiel in verteilten Systemen zum Einsatz kommen.

Neben der Unterscheidung, ob eine Bean lokal oder remote ansprechbar ist, gibt es drei verschiedene Bean Typen: Entity Beans, Session Beans und Message Driven Beans.

Entity Beans dienen der Anbindung des Systems an eine Datenbank. Sie modellieren die persistenten Daten der Anwendung und können so dem restlichen System die Datensätze der Datenbank zur Verfügung stellen. Damit entfallen für die Entwickler anderer Klassen die komplizierten Zugriffe auf die Datensätze, da die Entitiy Beans eine einfache und einheitliche Sicht auf diese bieten.

Session Beans modellieren den Systemablauf. Sie können weiterhin in *stateless* und *statefull* Session Beans unterteilt werden. Stateless, also zustandslos, bedeutet in diesem Zusammenhang, dass die Bean ihren Zustand nicht speichern kann. Statefull Session Beans unterstützen dies hingegen und können daher einen vorher unterbrochenen Zustand wieder aufnehmen. *Zustand* bezieht sich hierbei immer auf eine Benutzersession mit dem System.

Message Driven Beans implementieren eine asynchrone Kommunikation über das Java Messaging System. Durch diese Beans ist eine Kommunikation mit anderen Systemen, wie zum Beispiel einen Mailserver, möglich. Asynchronität bietet dabei den Vorteil, dass

nicht auf den Abschluss der Operation gewartet werden muss bevor der ursprüngliche Kontrollfluss fortgesetzt werden kann.[EJB]

4. Datenmanagement

Dieses Kapitel soll einen Überblick über die untersuchten Datenbankmodelle, verteilten Systeme, Datenmanagement-Interfaces und Möglichkeiten zur Metadatenverwaltung geben.

4.1. Datenbankmodelle

Datenbanken werden überall dort benötigt, wo Daten über einen längeren Zeitraum verfügbar gehalten werden müssen. In diesem Abschnitt wird ein Überblick über die verschiedenen Datenbankmodelle gegeben sowie eine anschließende Bewertung dieser auf Basis unterschiedlicher Anforderungen vorgenommen.

4.1.1. Das relationale Datenbankmodell

Bei dem relationalen Datenbankmodell handelt es sich um das älteste und auch bekannteste der hier betrachteten Datenbankmodelle. Erstmals vorgestellt wurde es in den frühen 1970er Jahren, und seitdem kontinuierlich weiterentwickelt, so dass relationale Datenbanken heutzutage sehr weit verbreitet sind und meist sehr effizient arbeiten. Relationale Datenbanken bestehen, anschaulich gesprochen, aus Tabellen, in deren Zeilen die Datensätze abgespeichert werden. Das Datenbankschema wird in diesem Fall durch die Zeilen definiert, über die der jeweilige Datentyp, Wertebereich und eine Reihe weiterer Eigenschaften (Schlüsselattribut, einzigartig, etc.) der zugehörigen Daten sowie eventuell eine semantische Bedeutung festgelegt werden. Datenbankoperationen werden in der Regel durch SQL (Structured Query Language) ausgeführt, die auf der Relationenalgebra basiert.
Um Beziehungen zwischen Datensätzen auszudrücken, ist es erforderlich, zusätzliche Tabellen mit der entsprechenden semantischen Bedeutung anzulegen, in denen Verweise auf die betreffenden Datensätze liegen. Die Konstistenz der Daten, auch im Fall konkurrierender Zugriffe, wird in der Regel durch eine Transaktionslogik gewährleistet. Innerhalb einer Transaktion werden dabei entweder alle Anweisungen dieser ausgeführt, oder aber gar keine. Des Weiteren wird gewährleistet, dass während der Bearbeitungszeit der

Transaktion keine Zugriffe auf die selben Daten durch transaktionsfremde Operationen ausgeführt werden, wenn dies zu inkonsistenten Zuständen führen könnte.
Ein Nachteil von SQL ist, dass es stark von der Zugriffslogik des umgebenden Rahmenprogramms abhängig ist. So eignet sich SQL gut für die Manipulation großer Datenmengen, jedoch im Falle des Zugriffs aus einem objektorientierten Rahmenprogramm, in dem es häufiger eine Vielzahl kleiner, lokaler Zugriffe gibt, kann es diese Stärken nicht ausspielen. Auch die Vielzahl der Beziehungen zwischen den Objekten lassen sich oft nur umständlich durch das relationale Datenbankmodell beschreiben. Als weitere Einschränkung gilt, dass in SQL rekursive Anfragen nicht möglich sind, weshalb an sich einfache rekursive Probleme im Rahmenprogramm in eine Vielzahl kleiner Anfragen aufgespalten werden müssen, was zusätzlichen Aufwand verursacht. Auch die Erstellung eines Query-Strings im Rahmenprogramm kann umständlich sein, da vom Programmierer in diesem Fall zusätzlich Kenntnisse der Datenbankstruktur sowie der Abfragesprache SQL verlangt werden.

4.1.2. Das objektrelationale Datenbankmodell

Bei dem objektrelationalen Datenbankmodell handelt es sich um eine Weiterentwicklung des relationalen Modells, das insbesondere die Schwachstellen von SQL in Bezug auf objektorientierte Rahmenprogramme beheben soll. So entfällt hier die Erstellung expliziter SQL-Abfragen im Rahmenprogramm, stattdessen wird einmalig eine Abbildung der Objektattribute auf die Datenbankattribute definiert. Dies erfolgt in der Regel über eine XML-Datei. Bei schreibendem Zugriff auf die Datenbank werden die Objektinformationen automatisch ausgelesen und deren Inhalt in den korrespondierenden Datenbankattributen gespeichert, der lesende Zugriff erfolgt auf ähnliche Art und Weise. Oft kann das Datenbankschema sogar automatisch aus den Quellcode-Dateien des Programms generiert werden, so dass der Programmierer die Abbildung von Objekt- auf Datenbankattribute nicht selbst vornehmen muss.
Da es sich bei objektrelationalen Datenbanken im eigentlichen Sinne auch um relationale Datenbanken handelt, unterstützen sie weiterhin SQL. Dies trägt dazu bei, dass diese Datenbanken sehr weit verbreitet sind, da sie somit ebenfalls in Anwendungen genutzt werden können, die lediglich SQL für Datenbankzugriffe nutzen. Des Weiteren verfügen sie oft auch über zusätzliche Sprachelemente, die rekursive Anfragen ohne Rückgriff auf Rahmenprogrammfunktionalität ermöglichen, was in SQL sonst nicht möglich ist.

PostgreSQL

PostgreSQL ist ein objektrelationales Datenbanksystem, das sich gut für die Speicherung von strukturierten Daten mit festem Schema anbietet. PostgreSQL existiert für verschie-

dene Plattformen, insbesondere auch für Linux und Windows und lässt sich mittels JDBC einfach in Java-Anwendungen integrieren. In den aktuellen PostgreSQL Versionen (9.0 und größer) wird auch Replikation, also die verteilte Speicherung und Synchronisation derselben Daten, unterstützt. Dies kann dazu genutzt werden die Skalierbarkeit des Gesamtsystems zu verbessern. Ein für die Projektgruppe wichtiger Vorteil bei der Auswahl von PostgreSQL als eingesetzte Datenbanklösung ist die freie Verfügbarkeit, so dass keinerlei Einschränkung bei der Nutzung im Vergleich zu anderen Lösungen vorliegt.[pos]

4.1.3. Das objektorientierte Datenbankmodell

Das objektorientierte Datenbankmodell lässt sich vereinfacht als „Persistierung" von Datenobjekten beschreiben. Anders als bei objektrelationalen Datenbanksystemen erfolgt keine automatische Abbildung der Datenfelder eines Objekts auf das Schema der Datenbank, ein solches Schema existiert hier nicht einmal. Eine Konsequenz aus dem Fehlen eines Datenbankschemas ist, dass ein Zugriff auf die Daten mittels SQL nicht mehr möglich ist. Zwar existiert mit OQL eine an SQL angelehnte Abfragesprache, in der Regel erfolgt der Zugriff auf Daten in einer objektorientierten Datenbank jedoch mittels Vorlagenobjekten (template objects) oder mittels Navigation. Vorlagenobjekte sind dabei Objekte des gleichen Typs wie das gesuchte Objekt (oder eines übergeordneten Typs, Vererbung wird in der Regel unterstützt), in dem bestimmte Datenfelder auf einen vorgegebenen Wert gesetzt sind, auf den die in der Datenbank gespeicherten Objekte überprüft werden. Während einfache Vorlagenobjekte nur auf Gleichheit prüfen können, lassen sich beispielsweise durch Übergabe mehrerer Objekte oder geeigneter Operatoren an die Datenbank auch Intervalle überprüfen. Der Zugriff mittels Navigation schließlich ist die natürlichste Art auf gespeicherte Daten zuzugreifen. Hierbei werden die in der Datenbank gespeicherten Objekte wie gewöhnliche Objekte in einem objektorientierten Rahmenprogramm über Zeiger bzw. Referenzen angesprochen. Von bereits geladenen Objekten referenzierte Objekte, die sich jedoch noch in der Datenbank befinden, werden aus dieser dynamisch nachgeladen. Der Einsatz von objektorientierten Datenbanken bietet sich überall da an, wo sehr unterschiedliche oder auch veränderliche Datenobjekte gespeichert werden müssen, für die sich nicht ohne weiteres ein einheitliches Schema finden lässt. Auch bei stark verzeigerten Daten, wie z.B. bestimmten Graphen, auf die oft mittels Navigation zugegriffen wird, empfiehlt sich der Einsatz einer objektrelationalen Datenbank. Für strukturierte Informationen sollte man hingegen besser auf objektrelationale Datenbanken zurückgreifen, da diese bei großen Datenmengen meist deutlich besser skalieren.[odb]

4.1.4. Das dokumentenorientierte Datenbankmodell

Ähnlich wie objektorientierte Systeme gehören auch dokumentenorientierte Datenbanken zu den sogenannten NoSQL-Datenbanken. Die Datenbankeinträge werden hierbei als Dokumente bezeichnet und bestehen aus Schlüssel-Wert-Paaren. Die Werte können, neben einfachen Datentypen wie Zahlen und Zeichenketten, auch Listen oder ganze Dokumente sein. Zudem ist die Struktur eines Dokuments nicht über ein fixes Schema festgelegt und kann jederzeit geändert werden.

MongoDB

Bei MongoDB handelt es sich um eine plattformunabhängige dokumentenorientierte Datenbank. Durch die Abwesenheit eines festen Schemas ist es ohne Probleme möglich weitere Metadaten zu einzelnen Dokumenten hinzuzufügen und später dann auch für Datenbankanfragen verwenden zu können. Lassen sich Daten jedoch nicht in ein Dokument einbetten, weil sie z.B. von mehreren Dokumenten referenziert werden, müssen Verknüpfungen von Hand realisiert werden, da es kein JOIN oder andere, einfache Möglichkeiten der direkten Referenzierung gibt. Deswegen wird MongoDB im Rahmen der Projektgruppe nur zur Metadatenverwaltung eingesetzt, während alle anderen Daten in einer objektrelationalen Datenbank vorrätig gehalten werden.[mon]

4.2. Verteilte Dateisysteme

Ein verteiltes Dateisystem kommt meistens auf verteilten Systemen zum Einsatz. Dazu zählen beispielsweise Grids oder ähnliche Systeme. Verteilte Dateisysteme sind im Gegensatz zu lokalen Dateisystemen über viele Knoten verstreut. Einige Rechnerknoten nehmen dabei spezielle Funktionen zur Verwaltung des Dateisystems ein. Global werden die Speicherkapazitäten der einzelnen Knoten jedoch virtuell zu einer großen Festplatte vereint. Der Anwender sieht nur einen großen Speicherplatz, ähnlich zu einem lokalen Dateisystem. Das lokale Dateisystem beschränkt sich auf ein lokales Speichermedium zum Beispiel einer Festplatte. Es verwaltet die Festplatte für den Benutzer und stellt ihm eine einfache Sicht auf die Ressource zur Verfügung. Bei verteilten Systemen müssen weitere Punkte beachtet werden wie zum Beispiel parallele simultane Zugriffe auf Dateien oder der Ausfall von Speicherknoten.

Hauptsächlich werden verteilte Dateisysteme dort eingesetzt wo große Kapazitäten und massive parallele Zugriffe bei hoher Skalierbarkeit und stabilem Datendurchsatz gefordert

werden. Lokale Pendants sind dafür nicht ausgelegt und können nicht in einem verteilten Umfeld eingesetzt werden.

Unsere Projektgruppe hat es sich zur Aufgabe gemacht eine skalierbare Datenstruktur für verteilte Systeme zu entwickeln, die sich vieler Prinzipien der verteilten Dateisysteme bedient. Daher haben wir uns für eine Analyse der bekanntesten verteilten Dateisysteme entschieden, um mögliche Konzepte aufgreifen zu können. Bereiche wie die Metadatenverwaltung oder die hohe Abstraktionsschicht der Anwendersicht fließen in unsere Software ein und werden an vielen Stellen aufgegriffen.

Viele der verteilten Dateisysteme haben einige gemeinsame Konzepte, die in den nachfolgenden Kapiteln über XtreemFS 4.2.1 und Lustre 4.2.2 aufgegriffen werden. Dazu zählt zum Beispiel die Trennung der Metadatenverwaltung von der Datenhaltung. Die Metadaten werden in einem separaten System, das meistens aus einem Master-Slave System besteht, verwaltet. Dazu könnte zum Beispiel eine verteilte Datenbank zum Einsatz kommen, um die Metadatenverwaltung durch Replikation abzusichern. Häufig gibt es einen Master auf dem die Daten geschrieben werden und beliebig viele Slaves von denen nur gelesen werden darf. Falls der Masterknoten ausfällt wird ein aktueller Slaveknoten zum Master hochgestuft. Die Implementierung der Datenhaltung erfolgt über eigenständige Verfahren. Dabei werden die Daten meistens in Teile aufgeteilt und auf viele Speicherknoten verteilt. Durch die Replikation der Pakete betrifft der Ausfall eines Knotes eine konkrete Datei häufig nicht, da sie mit Hilfe der Replikate zum Client übertragen werden kann. Durch spezielle Optimierungsalgorithmen werden die Replikate effizient verteilt, so dass ein häufiger Zugriff auf eine Datei die Häufigkeit der zu erzeugenden Replikate und den Ort bestimmt.

4.2.1. XtreemFS

XtreemFS ist ein objektbasiertes-, verteiltes Dateisystem für Grids. Es kann Objekte replizieren, um eine höhere Ausfallsicherheit zu gewährleisten. Das Dateisystem hat einen Cache für Daten und Metadaten damit die Latenz für Zugriffe sinkt. Zusätzlich kann es Verbindungen mit Hilfe von SSL verschlüsseln, damit eine sicherere Verwendung über unsichere Netzwerke möglich wird.

In der aktuellen Version ermöglicht das verteilte Dateisystem paralleles lesen und schreiben von Daten über einen nativen Windows und Linux Client. Da die Daten objektbasiert gespeichert werden, sind Daten und Metadaten voneinander getrennt. Durch eine Abstraktionsschicht werden die zugrundeliegenden Strukturen dem Benutzer verborgen und im Client wie ein normales Dateisystem angezeigt. Sowohl die Daten als auch die Metadaten können auf mehrere Nodes verteilt werden, um einen hohen Datendurchsatz zu ermöglichen. Durch spezielle Algorithmen wird sichergestellt, dass die Daten geographisch nahe zum Benutzer abgelegt werden. Bei hohem Bedarf spezieller Daten, können

diese lokal repliziert werden. Dieses Verfahren ermöglicht außerdem das Weiterarbeiten auf den Daten bei einem Netzwerkausfall. Durch die getrennte Metadatenhaltung erfolgt eine Suche nach Dateien über den zentralen Metadatenserver. Diese Suche funktioniert so effizient und einfach, wie eine simple Datenbankabfrage.

Weitere Vorteile der Architektur von XtreemFS sind die Kompatiblität zu POSIX und die nicht benötigte Spezialhardware. XtreemFS läuft auf Consumer Hardware und bietet durch die POSIX Kompatiblität die Möglichkeit bisherige Anwenundungen uneingeschränkt ohne Anpassungen laufen zu lassen.[HCK+08]

Client/Access Layer

Der Access Layer stellt die Schnittstelle zwischen den Benutzerprozessen und dem Dateisystem dar. Sie ermöglicht dem Client den Zugriff auf Dateien und Ordner des verteilten Dateisystems. Alle Schnittstellen sind wie bereits erwähnt POSIX-kompatibel und können dadurch leicht in bestehende Komponenten integriert werden.

Die Aufgabe des Access Layers besteht darin, die vom Client übertragenen Aufrufe, in für das „Object Storage Device-“ und den „Metadata and Replica Catalog“ verständliche Kommandos zu übersetzen. Es handelt sich um eine Vermittlungsschicht zwischen dem Client und dem verteilten Dateisystem. Durch diese Schicht kann der Benutzer XtreemFS wie ein lokales Dateisystem einbinden. Zusätzlich bietet der Layer Verwaltungsmöglichkeiten für das Dateisystem an, um es administrieren zu können. [HCK+08]

OSD - „Object Storage Device“

Das Object Storage Device ist für die Speicherung des Inhaltes der Dateien verantwortlich. Im Dateisystem werden Dateien durch eines oder beliebig viele Objekte repräsentiert. Um eine höhere parallele Lese-/Schreibrate zu erzielen, können diese Objekte auf verschiedene Serverknoten verteilt werden. Auch eine partielle Aufteilung ermöglicht das Object Storage Device, falls nur einige Teile häufig benötigt werden. Dadurch können Ressourcen eingespart werden und gleichzeitig die Performance erhöht werden. Diese Objekte werden auf beliebige weitere Server verteilt, die als Object Storage Device konfiguriert sind. Replikate werden immer geographisch nahe zum Benutzer abgelegt und, für den Fall, dass ein Knoten nicht erreichbar ist dupliziert. Das Object Storage Device muss die einheitliche Konsistenz der Replikate garantieren. Daher werden alle Zugriffe auf die Objekte überwacht und geregelt.

MRC - „Metadata and Replica Catalog“

Der „Metadata and Replica Catalog“ ist zur Verwaltung der Metadaten der Dateien und Ordner im Dateisystem. Um auch hier eine größere Bandbreite an parallelen Anfragen bedienen zu können, kann es mehrere dieser Metadata and Replica Kataloge geben. Der Dienst speichert die Metadaten in einer Datenbank und funktioniert nach dem Master-/Slave Prinzip. Es kann pro Dateisystem nur einen Master geben und beliebig viele Slaves. Eine Änderung am Master wird automatisch auf die Slaves übertragen. Daher haben alle Knoten immer einen konsistenten Datenbestand. Um auf die Daten zugreifen zu können, stellt der Dienst Interfaces bereit.

RMS - „Replica Management Service“

Im Gegensatz zum Metadatendienst entscheidet der „Replica Management Service“ wann ein neues Replikat erstellt werden muss oder wann es entfernt werden kann. Der Dienst überwacht die Transaktionen der Benutzer und erzeugt durch Optimierungsverfahren in der Nähe der Benutzer Replikate von Dateien auf die häufig zugegriffen wird. Wenn Knoten ausfallen oder Inkonsistenzen drohen, greift der Replica Management Service ein und führt Korrekturmaßnahmen durch. Der Dienst legt die so genannte „striping-policy“ fest und bestimmt damit wie oft repliziert wird. Außerdem muss der Dienst sich alle erzeugten stripes merken, um sie verwalten zu können.

Erweiterbarkeit

XtreemFS lässt sich durch diverse Schnittstellen wie zum Beispiel CORBA beliebig erweitern. Module können sowohl für C++ als auch für Java geschrieben werden. Damit eignet sich das Dateisystem hervorragend für eigene Erweiterungen. Alle Schnittstellen und Protokolle sind offen und dokumentiert. Für jede Interaktion mit dem Dateisystem, zum Beispiel Datei lesen, existieren Sequenzdiagramme, die den genauen Ablauf wiedergeben.

4.2.2. Lustre

Bei Lustre handelt es sich, ähnlich wie bei XtreemFS, um ein hochgradig paralleles Dateisystem für verteilte Systeme wie zum Beispiel Grids. Es kann weit über 10000 Nodes bedienen und mindestens ebenso viele Clients abarbeiten. Es verwaltet Kapazitäten bis zu einigen Petabytes und wird auf über 50% der Top 30 Supercomputer eingesetzt. Lus-

tre bietet seinen Nutzern Datenraten von bis zu 100 Gigabytes pro Sekunde. Wie auch XtreemFS ist Lustre hoch skalierbar und damit fast beliebig erweiterbar. [lus]

Architektur

Wie auch bei XtreemFS muss Lustre einen Metadatenserver pro Dateisystem haben und kann einen oder beliebig viele Object Storage Server verwalten. Auch auf Lustre greifen Clients über spezielle Software zu, um das Dateisystem nutzen zu können.

4.2.3. Amazon S3 - „Simple Storage Service“

Das letzte Beispiel umfasst das Amazon S3 Dateisystem: „Simple Storage Service“. Es speichert Daten in so genannten Buckets. Diese Buckets werden ähnlich wie normale Ordner auf einem lokalen Dateisystem verwendet. Ein Bucket speichert unbegrenzt viele Datenobjekte. Ein solches Objekt besteht aus einem binären Datenobjekt, dem Namen und seinen Metadaten. Für die Benutzung von Amazons S3 fallen Gebühren an. Diese Gebühren werden für die belegte Kapazität in Gigabyte pro Monat, dem Datentransfer für Up- und Download erhoben und für die Operationen auf Dateien wie zum Beispiel get, put und list. Amazon bietet seinen Kunden zur Sicherheit und Verschlüsselung der Daten mittels asymmetrischer Verfahren an. Um die globale Verfügbarkeit zu gewährleisten spiegelt Amazon nach eigenen Angaben die Daten parallel in mehreren Datenzentren.[s3F]

4.3. Datenmanagement-Interfaces

Neben Cloud-Computing zur Analyse großer Datenmengen ist Cloud-Storage zur Speicherung dieser Daten ebenfalls ein wichtiges Thema. Cloud-Storage bietet DaaS, damit die Daten in einem virtuellen Pool gespeichert werden können. Wichtig für jede Implementierung von DaaS ist die Unterstützung von Standard-Protokollen wie iSCSI für den blockbasierten Datenzugriff sowie von CIFS, NFS sowie WEBDAV zur Implementierung von Netzwerkdateisystemen (siehe Abb. 4.1). Außerdem bietet Cloud-Storage Backup-Systeme und Möglichkeiten zur Synchronisierung von Daten auf unterschiedlichen Geräten. Des Weiteren stellt es oft eine Schnittstelle zum Cloud-Computing zur Verfügung. Der Zugriff auf in der Cloud gespeicherte Objekte erfolgt in der Regel über CRUD-Operationen. Dabei ist es auch möglich, mehrere Objekte in einem Container zu gruppieren und diese anschließend gemeinsam zu manipulieren (siehe Abb. 4.2).

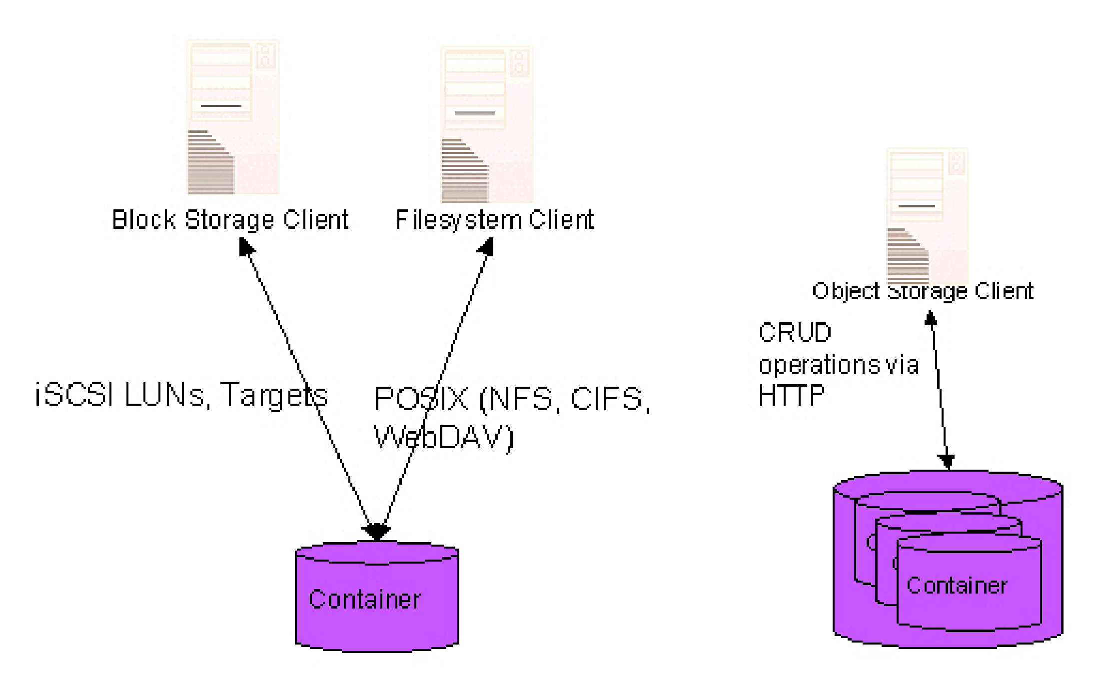


Abbildung 4.1.: Standards für Datenzugriffs-Interfaces (Entnommen aus [SNI11])

Abbildung 4.2.: Interfaces für Objekt-Zugriff

4.3.1. CDMI

Bei CDMI [1] handelt es sich um eine Entwicklung der SNIA, die die Einfachheit von Cloud Storage erhält. CDMI definiert ein funktionales Interface, dass es Entwicklern ermöglicht, aus ihrer Anwendung heraus Daten in einer Cloud zu manipulieren. Die von CDMI genutzten Metadaten gliedern sich dabei in drei Bereiche: User-Metadaten, die genutzt werden, um Daten-Objekte und Container zu finden, Storage-System-Metadaten, die Zugriffe, Zugriffsrechte und Statistiken betreffen sowie Data-System-Metadaten, die die Operationen des Data-Service kontrollieren (siehe Abschnitt 4.3).

CDMI ermöglicht es, Dateien in der Cloud zu erzeugen, zu verändern, anzufordern und zu löschen. Außerdem stellt es die folgende Funktionalität bereit:

- CDMI ermöglicht es einem Client, angebotene Ressourcen und Dienste in der Cloud aufzufinden.
- CDMI verwaltet die Container und darin enthaltene Daten.

[1] `http://cdmi.sniacloud.com/`, CDMI-Dokumentation, zuletzt abgerufen am 20.11.2011

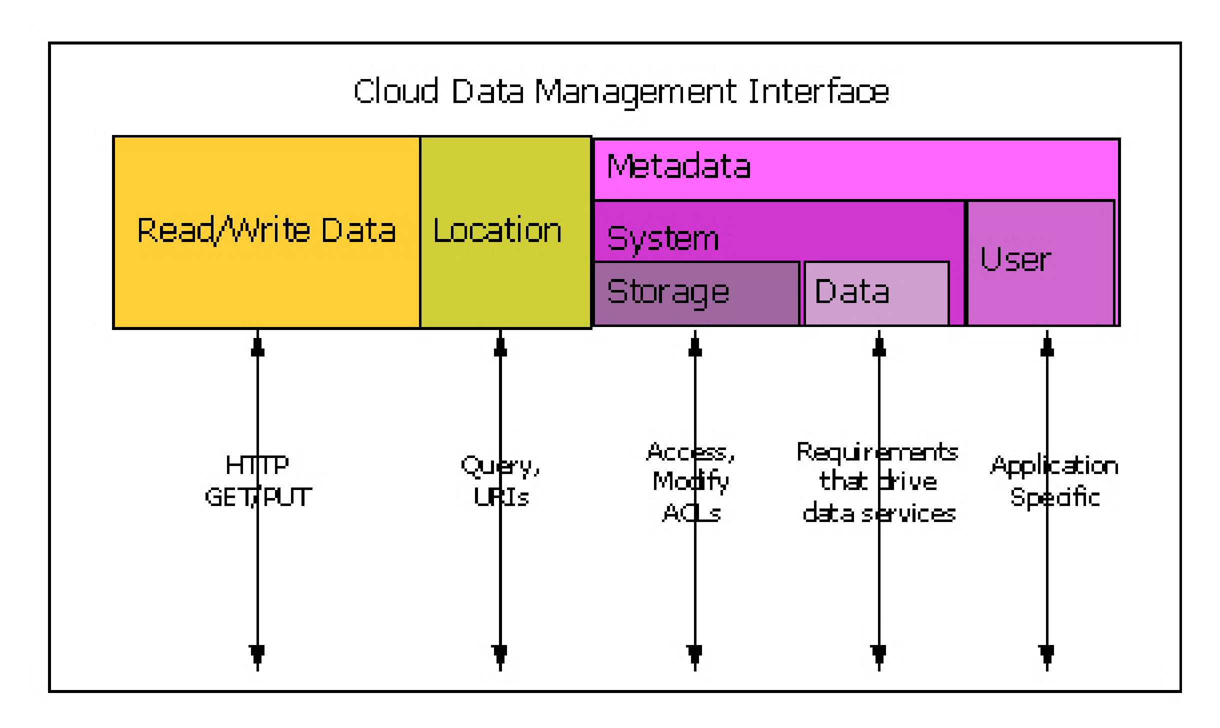


Abbildung 4.3.: Das SIRDM im Cloud-Storage (Entnommen aus [SNI11])

- CDMI ermöglicht die Verknüpfung von Containern und Datenobjekten mit Metadaten.

Die beiden wichtigsten Interfaces von CDMI sind das Data-Path-Interface und das Control-Path-Interface. Das Data-Path-Interface ist dabei für die Speicherung und das Anfordern der Daten zuständig, das Control-Path-Interface für die Datenverwaltung. CDMI verwendet dabei hauptsächlich RESTful HTTP und JSON als Protokolle. Auf der Website der SNIA steht eine Beispielimplementierung des CDMI-Servers zum Download zur Verfügung [2]. Außerdem steht an anderer Stelle [3] ein frei verfügbarer CDMI-Client zum Download bereit.
Die Kommunikation zwischen diesem Client und dem Server erfolgt mittels RESTful HTTP. Dabei sendet der Client die entsprechenden Anweisungen (z.B. GET, POST, PUT, DELETE) an den Server, dieser antwortet dann mittels HTTP-Statuscode und sendet eventuell angeforderte Daten bzw. Metadaten zurück an den Client (siehe Abb. 4.4).

[2] `http://www.snia.org/forums/csi/programs/CDMIportal`, CDMI-Server, zuletzt abgerufen am 20.11.2011
[3] `https://github.com/livenson/libcdmi-java`, CDMI-Client, zuletzt abgerufen am 20.11.2011

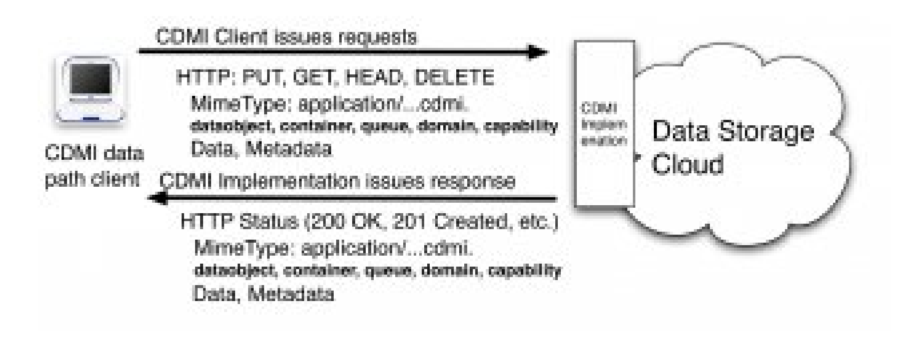


Abbildung 4.4.: CDMI-Verbindung (Entnommen aus [Car11])

4.4. Metadatenverwaltung

Bei Metadaten handelt es sich um Informationen, die sich das Dateisystem zu den von ihm verwalteten Dateien merkt. Sie beschreiben bestimmte Eigenschaften der Dateien. Typische Metadaten sind beispielsweise die Dateigröße und das Datum der letzten Änderung einer Datei, aber auch weniger offensichtliche Dinge wie Zugriffsberechtigungen oder der Dateipfad sind strenggenommen Metadaten. Auch wenn Metadaten prinzipiell beliebig definiert werden können, werden Dateien selbst oder deren Inhalt in der Regel nicht als Metadaten bezeichnet, obwohl diese als vollständige Beschreibung ihrer selbst betrachten werden könnte. Die Metadatenverwaltung beschäftigt sich mit typischen Operationen, wie dem Anlegen, Aktualisieren, Löschen und insbesondere dem Suchen von Metadaten.

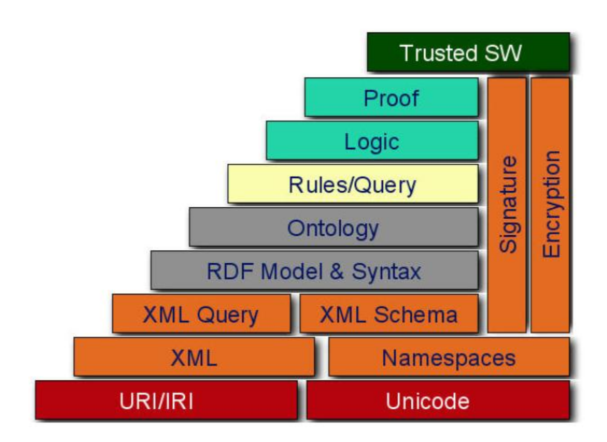


Abbildung 4.5.: Schichtenmodell des „Semantischen Webs“ (herausgenommen aus [semb])

4.4.1. Einschränkungen von XML

Obwohl XML ein mächtiges Sprachmittel ist, um Daten zu strukturieren, reicht es alleine für die Beschreibung von Metadaten nicht aus. Zunächst einmal besteht ein Problem bezüglich der Eindeutigkeit der Metadaten, da man in der Lage ist, eine Information mit verschiedenen Tags zu beschreiben. Außerdem gibt XML selbst keine Konventionen bezüglich der Art der Beschreibung von Metadaten vor. Drittens, das Problem des Parsens hängt von der jeweiligen Struktur (geschrieben in DTD oder XML Schema) ab. Wenn man nun möchte, dass eine Ressource mittels Metadaten Informationen über sich bereitstellt, die darüber hinaus noch in einem maschinenlesbaren Format vorliegen sollen, werden die folgenden Dinge benötigt:

- Eine eindeutige Bezeichnung der Ressourcen (URIs).
- Ein gemeinsames Datenmodell zur Spezifikation der Metadaten (RDF).
- Ein gemeinsames Vokabular (Ontologie OWL). [sema]

4.4.2. Der RDF-Sprachstandard

RDF (Resource Description Framework) ist eine Sprache, die die Beschreibung von Ressourcen erlaubt. Eine Ressource ist hierbei nicht nur eine Webseite, eine Datei oder ein anderes (virtuelles) Objekt, sondern kann auch eine Person, eine Firma, oder ein real existierender Gegenstand sein. Metadaten in RDF bestehen aus einer Menge von Aussagen (Statements). Jedes Statement ist ein Tripel aus Subjekt, Prädikat und Objekt. Ein einfaches Beispiel dazu ist „Student besucht Vorlesung". Dabei ist „Student" das Subjekt, „besucht" das Prädikat, und „Vorlesung" das Objekt. Für RDF gibt es mehrere Notationen bzw. Visualisierungen, zum Beispiel Graphen, RDF/XML, N3, Turtle und RXR. Dabei ist RDF/XML das „offizielle" Metadaten-Format des World Wide Web Consortiums(W3C). Zum besseren Verständnis werden die Tripel (S,P,O) als gerichtete Graphen mit Kantenbeschriftung betrachtet. Dabei sind das Subjekt und das Objekt die Knoten des Graphen, das Prädikat die Beschriftung der Kanten.
So lässt sich auch dann leicht ausdrücken, dass eine Person eine bestimmte Telefonnummer oder Email-Adresse hat, wenn die beiden Informationen (Email und Telefonnummer) an unterschiedlichen Orten gespeichert werden. Das RDF-Datenmodell erlaubt es, diese Informationen gemeinsam zu nutzen und zu kombinieren. Eine Anwendung kann die Graphen, welche die selbe URI beschreiben, zusammenführen. Dabei entsteht ein RDF Graph, der neue Informationen liefert. Die Graphen sind für Menschen gut verständlich und sind für die Komposition verteilter Informationen geeignet.
Für die eindeutige Beschreibung einer Ressource dient der URI (Uniform Ressource Identifier). Dieser kann als ein Oberbegriff zur URL(Uniform Ressource Locator) verstanden

werden. URIs können nicht nur die Adresse einer Webseite, sondern auch bestimmte Person oder Objekte aus dem realen Leben eindeutig identifizieren, unabhängig davon, ob sie online verfügbar sind oder nicht.
In RDF entsprechen den Objekten bestimmte Datenwerte, die als Literale dargestellt werden. Der Wert eines Literals ist eine Zeichenkette und hat reservierte Bezeichner für einen bestimmten Datentyp. In der Graph-Darstellung werden die Literale als Vierecke dargestellt.
Die W3C empfiehlt das RDF/XML-Format, welches eine XML-Notation für die Beschreibung der Tripel erlaubt. Das Beispiel mit der Telefonnummer lässt sich in RDF/XML-Notation etwa so beschreiben:

```
<rdf:Description rdf:about="http://www.tu-dortmund.de/studierende/~Erika_Mustermann">
<term:telefon>+49231123456</term:telefon>
</rdf:Description>
```

Das Attribut rdf:about referenziert einen URI einer bestehenden Ressource. Eine neue Ressource kann mit Hilfe des Konstrukts rdf:ID definiert werden. In RDF/XML-Syntax ist es erlaubt, mehrere Tripel ineinander zu verschachteln und zu kombinieren. Dabei entsteht ein für Menschen unter Umständen schwer lesbares Metadaten-Dokument. Allerdings kann man die Tripel in einfacherer Syntax mit Hilfe von N3 (Notation 3) darstellen. Dies ist bezüglich der Ausdrucksmächtigkeit gleichwertig zu RDF/XML, aber einfacher und für Menschen verständlicher. Informationen in N3-Notation sind Sätze, die natürlicher Sprache ähneln:

```
<#Erika><#studiert><#Informatik>.
```

Jeder Teil eines solchen Satzes, also Subjekt, Prädikat oder Objekt, ist entweder ein URI oder ein Literal mit seinem Datenwert. Das Schreiben von Daten in ein RDF-Dokument bezeichnet man, unabhängig von der gewählten Notation, als „Serialisierung“. Obwohl diese Daten in einem maschinenlesbaren Format vorliegen, haben die darin enthaltenen Informationen für einen Computer zunächst keinerlei Bedeutung. Um eine solche Bedeutung zu spezifizieren und einfache Ontologien zu beschreiben kann das RDF-Schema verwendet werden.
Das RDF-Schema ermöglicht terminologisches Wissen über die in einem Vokabular verwendeten Begriffe zu spezifizieren. In RDFS sind die Begriffe „:Resource“ und „:Class“ definiert, wobei alles, was als URI beschrieben werden kann, unter den Begriff „:Resource“ fällt.

rdfs:Resource ist die allgemeinste Klasse in RDFS. Jeder in RDF beschriebener URI ist eine Instanz dieser Klasse.

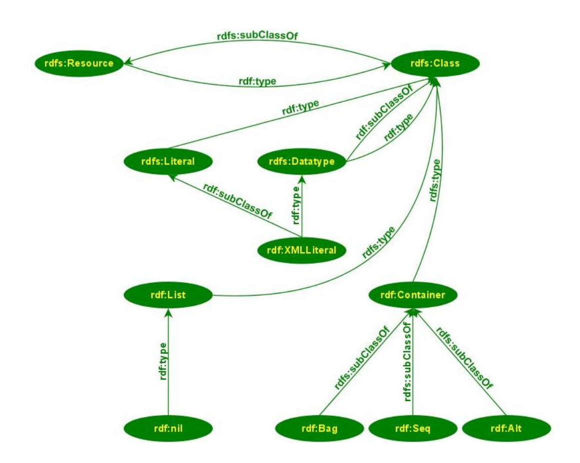


Abbildung 4.6.: RDF-Schema

rdfs:Class ist die Oberklasse aller Klassen in RDFS. Das heißt, dass jede definierte Klasse die „:type“-Eigenschaft von „:Class“ hat. Das Konstrukt „:type“ erlaubt es, die Zugehörigkeit zu einer Klasse zu definieren.

rdfs:Property definiert die Eigenschaften eines URI.

rdfs:subClassOf und **rdfs:subPropertyOf** dienen der Vererbung von Klassen und Eigenschaften.

Die Abbildung 4.6 zeigt die Zugehörigkeit und Eigenschaften für die Konstrukte im RDF-Schema. Hier ist beispielsweise „rdfs:Class“ eine Unterklasse von „rdfs:Resource“ und gleichzeitig ist „rdfs:Resource“ eine Instanz von „rdfs:Class“.

Zwar ist RDF-Schema ein mächtiges Werkzeug, dennoch bleiben einige Probleme bestehen: So kann die Äquivalenz zweier Klassen oder Eigenschaften nicht automatisch festgestellt werden. Für dieses und weitere Probleme ist OWL besser geeignet. [rdf]

4.4.3. OWL

OWL (Web Ontology Language) ist eine Repräsentationssprache zum Erstellen von Ontologien. OWL basiert auf RDF und RDFS und ist in XML-Syntax formulierbar. Sie bietet die Fähigkeit logische Operationen auf Klassen auszuführen und Enumerationen zu bilden. OWL hat drei verschiedene Sprachebenen: OWL Lite, OWL DL, OWL Full, die unterschiedlich mächtig sind.

OWL Lite ist eine echte Teilsprache von OWL DL. OWL DL ist wiederum eine echte Teilsprache von OWL Full. OWL Lite bietet einfache Klassifikationshierarchien und Ein-

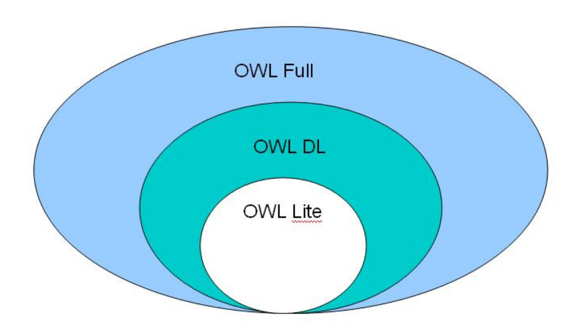


Abbildung 4.7.: Die OWL-Versionen.

schränkungen, ist entscheidbar und hat eine geringe Ausdrucksstärke. OWL DL bietet die größtmögliche Ausdrucksstärke, bei der die Entscheidbarkeit immer noch gesichert ist und wird von aktuellen Softwarewerkzeugen nahezu vollständig unterstützt. OWL Full hingegen hat maximale Ausdrucksstärke, ist aber nicht immer entscheidbar. Als einzige Variante von OWL enthält es alle Sprachkonstrukte von RDFS, allerdings wird es von aktuellen Softwarewerkzeugen nur bedingt unterstützt. Wenn man eine Ontologie entwickelt, empfiehlt es sich Gedanken darüber zu machen, welche Sprachvariante den eigenen Bedürfnissen am besten entspricht.[owl]

4.4.4. SPARQL

Wie wir gesehen haben, ermöglicht RDF Information zu strukturieren und miteinander zu verknüpfen. Aus RDFS-Dokumenten können andere Dokumente abgeleitet werden und OWL-Wissensbasen können neue OWL-Axiome implizieren. Um Anfragen an RDF-Dokumente zu spezifizieren wurde die Anfragesprache SPARQL entwickelt. Im Januar 2008 ist SPARQL zum „offiziellen" Standard des W3C geworden. SPARQL orientiert sich an der SQL-Syntax, dennoch gibt es einige Unterschiede. Ähnlich wie SQL basiert SPARQL auf einfachen Bausteinen, die sich zu komplexeren Anfragen zusammenfügen lassen, um die Ausgabe zusätzlich zu filtern und zu formatieren. In SPARQL werden die RDF-Graphen als eine Liste von Aussagen angesehen. Mit Hilfe von SPARQL-Anfragen können bestimmte Ressourcen ausgewählt werden, die vorgegebene Eigenschaften erfüllen.
Folgendes Beispiel enthält eine kleine Liste von Büchern in RDF-Format:

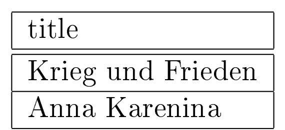
```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:books="http://www.example.org/schemas/books#"
xmlns:dc="http://purl.org/dc/elements/1.1/" >
<books:Book dc:title="Krieg und Frieden"
dc:author="Leo Tolstoy"
```

```
books:price="30" />
<books:Book dc:title="Anna Karenina"
dc:author="Leo Tolstoy"
books:price="25" />
<books:Book dc:title="Schuld und Sühne"
dc:author="Fjodor Dostojewski"
books:price="25" />
</rdf:RDF>
```

Mit der folgenden SPARQL-Anfrage lassen sich nun alle Buchtitel von Leo Tolstoy abfragen:

```
PREFIX dc <http://purl.org/dc/elements/1.1/>
PREFIX books <http://www.example.org/schemas/books#>
SELECT  ?title
WHERE { ?book dc:title ?title .
?book dc:author "Leo Tolstoy" }
```

Diese Anfrage enthält drei Schlüsselwörter PREFIX, SELECT und WHERE. Mit PREFIX ist der Namensraum deklariert. In unserem Beispiel sind es „dc“ und „books“. SELECT ist für das Ausgabenformat verantwortlich. Es liefert alle ?title, für die die Bedingungen in der WHERE-Aussage erfüllt ist. Die folgende Tabelle zeigt die Ergebnisse dieser Anfrage.



title
Krieg und Frieden
Anna Karenina

Tabelle 4.1.: Ergebnis der Anfrage auf die Buchliste

SPARQL erlaubt, einfache Graph-Muster zu kombinieren. Damit besteht die Möglichkeit alternative Muster anzugeben oder bestimmte Bedingungen nur an einen Teil der Anfrage zu stellen. [spa]

5. Architektur und Entwurf

Der Projektauftrag zur Projektgruppe Scalable Virtual Data Infrastructure sieht vor, dass eine Abstraktionsschicht alle den Ressourcen vorgeschaltet werden soll, die im Forschungsalltag verwendet werden. Um mit dieser Schicht zu kommunizieren, sollte ein beliebiger Client genutzt werden können. Mit dieser Anforderung wurde bereits der Grundstein für eine Client-Server-Architektur gelegt.

Diverse Brain-Stormings in der Gruppe brachten auch andere Ansätze hervor, zum Beispiel in denen Clients gleichzeitig Server sind. Auch die klassischen Varianten, in denen Server- und Client-Funktionalitäten voneinander getrennt sind, wurden diskutiert. Im Folgenden sollen die in Betracht gezogenen Architekturmodelle beschrieben und der Weg zur aktuellen Architektur darlegt werden.

5.1. Der Peer-To-Peer-Ansatz

Dieser Ansatz zeichnet sich dadurch aus, dass alle Systeme, die an der Kommunikation teilnehmen, gleich geartet und gleich berechtigt sind. Ein jedes System erhält somit die Logik, die ein dedizierter Server haben müsste, und die Logik eines Clients. Somit wären die Informationen über ansprechbare Ressourcen, die Metadaten zu den physisch gespeicherten Dateien und das Rechtesystem auf alle Clients verteilt bzw. jeder Client hat alle Informationen lokal gespeichert. Um den Nachrichtenaufwand möglichst gering zu halten wurde angedacht eine logische Ringstruktur mit sogenannten Superknoten zu realisieren, an die sich die Clients wenden können, um Informationen für einen Dateitransfer zu erfragen. Durch diese Abwandlung wäre auch der erforderliche Speicherbedarf der hinter den Superknoten gelagerten Knoten gering, da diese immer nur Teile des gesamten Informationsportfolios lokal vorliegen haben. Des Weiteren hätte sich die Synchronisation der Datenbasis nur auf die Superknoten verlagert, da die ausführenden Knoten jede durchgeführte Änderung an einen Superknoten zurückgemeldet hätten. Dieser Superknoten hätte nun die Änderung an alle anderen Superknoten kommuniziert. Allerdings wurde schnell festgestellt, dass neben der Synchronisierung auch die Deadlockerkennung bzw. -vermeidung ein Problem ist, welches sich nicht so einfach lösen lässt. Eine weitere Problematik tritt zu Tage, wenn es um ein Rechtesystem geht. Eine verteilte Rechtedatenbank wäre schwierig aktuell zu halten. Ebenso gestalteten sich Anfragen an eine verteilte Datenbank für die Mitglieder der Projektgruppe als schwierig. Auch ein knapper Zeitplan,

und bezüglich dieser Problematiken fehlende Kenntnisse im Bereich der verteilten Systeme, ließen diesen vielversprechenden Ansatz im Vergleich mit den folgenden als schwierig realisierbar erscheinen.

5.2. Fat-Client und Server

Eine Konsolidierung der Funktionen der Superknoten im vorherigen Ansatz zeigte, dass auch ein Server mit eben diesen Funktionalitäten eines Superknoten zielführend sein könnte. In diesem Ansatz hat der Client alle Funktionen für den direkten Dateizugriff auf jede beliebige Ressource und der Server verwaltet die globalen Informationen über alle Metadaten und Benutzer exklusive der Logininformationen eines Benutzers auf einen externen Server. Dies sind Informationen, wie zum Beispiel die auf einer physischen Ressource befindlichen Dateien. Alternativ handelt es sich um die berechtigten Benutzer, die auf einen Informationssatz zugreifen dürfen. Dazu müssen diese Daten vom Server erfragt werden. Der Server besteht aus einem Modul für die Metadatenverwaltung, die Ressourcen- und Nutzerverwaltung und die Rechteverwaltung. Im Gegensatz dazu beinhaltet der Client die Module für den Dateizugriff. So fiele dem Client die ausführende Rolle und dem Server die verwaltende Rolle zu.

Dieser Ansatz hat den Vorteil, dass alle Clients direkt auf die Ressourcen zugreifen können, ohne über einen zentralen Server die Verbindung aufbauen zu müssen, der ggf. zu einem Flaschenhals werden könnte. So trägt jeder Client selbst dafür sorge, dass die Dateien korrekt übertragen werden und können sogar untereinander Dateien direkt austauschen.

5.3. Thin-Client und Server

Diese Client-Server-Architektur sieht einen weniger mächtigen Client und einen mächtigeren Server vor. Alle Aktionen, die ein Benutzer, der den Client bedient, durchführen möchte, müssen entweder durch den Server ausgeführt oder mit diesem abgesprochen werden. Der Server dient in diesem Design sowohl als Informations- als auch als Kontrollinstanz. Mittels Modulen für die Metadatenverwaltung, die Dateitransaktionen und dem Rechtemanagement werden die Anforderungen des Clients umgesetzt. Ist der Client am Server angemeldet und authentifiziert, kann ein Ausschnitt der Metadaten angefordert werden, auf die er berechtigt ist zu zugreifen. Damit bekommt der Benutzer eine Übersicht über alle ihm zur Verfügung stehenden Daten und deren Status. Den Download einer Datei kann der Benutzer direkt initiieren, da der Client ein Modul besitzt, welches es ihm ermöglicht eine Verbindung zu einer oder mehreren Ressourcen aufzubauen und Dateien herunterzuladen. Bevor der Download allerdings durchgeführt werden kann,

fragt der Client nach einem Metadaten-Update, um kürzliche Änderungen zu erfahren. Danach lädt der Client die Datei direkt von der jeweiligen Ressource ohne den Server zu belasten. Führt der Client auf den Daten nun Änderungen aus und möchte diese zurückspeichern, so wird der Server mit dem Update beauftragt. Der Server nimmt die Daten entgegen, baut eine Verbindung zur Ressource auf und überschreibt die alten Dateien auf der Ressource, mit der aktuellen Version. Danach werden die Metadaten für die Dateien aktualisiert. Damit es nicht zu Inkonsistenzen beim Download und Update der Dateien kommt, werden vor der Übertragung der Änderung die Daten gesperrt. Diese Sperrung verhindert einen doppelten Zugriff auf ein Datum.

5.4. Gewählter Ansatz

Der von den Mitgliedern der Projektgruppe 553 zuerst gewählte Ansatz war die in Abschnitt 5.3 beschriebene Client-Server-Architektur mit einem Thin-Client. Viele der beschriebenen Probleme dieses Ansatzes lassen sich durch die Verwendung eines Applikationsservers, der einen abstrahierten Datenbankzugriff, Funktionalitäten für die Verwendung von Webservices und einen Transaktionsmanager bietet, beheben. Durch einen Fehler des Architekturteams stellte sich allerdings erst recht spät heraus, dass der Server immer nur für einen Benutzer eine Verbindung zu einer Ressource aufbauen konnte und alle andere Benutzer, die Sitzung mitbenutzen mussten. Diese Tatsache war nicht akzeptabel, so dass sich die Projektgruppe dazu entschied den in Abschnitt 5.2 beschriebenen Ansatz des Fat-Clients weiterzuverfolgen.

5.5. Verwendetete Datenbanken und Server

Die Entscheidung für die PostgreSQL-Datenbank bietet dem Administrator manuelle Fehlerbehebungsmöglichkeiten mittels nativen SQL-Anweisungen. Des Weiteren ermöglicht uns PostgreSQL eine automatische Replikation der gespeicherten Informationen. Diese Eigenschaft war für dieses System, welches von seinen gespeicherten Informationen abhängig ist, ausschlaggebend. Die PostgreSQL-Datenbank speichert in der endgültigen Version der SVDI die relevanten Informationen über die Benutzer, wie Emailadresse, Sitzungschlüssel, Loginzeitpunkt und Benutzername. Zusätzlich sind in dieser Datenbank auch die Informationen über die verfügbaren Ressourcen hinterlegt.

Die Metadaten für die verwalteten Dateien werden in einer NoSQL Datenbank gehalten. Die Wahl fiel auf MongoDB, da es auf Grund des fehlenden MVCC[1] hohe Update-Raten verspricht und sich so für ständig aktualisierende Daten, wie die Metadaten,

[1] Multiversion Concurrency Control

empfiehlt. Des Weiteren nimmt MongoDB es dem Programmierer ab sich im Falle von Konflikten, entstanden durch Transaktionen, um deren Auflösung zu kümmern. Zusätzlich ermöglicht MongoDB das Erstellen von einfachen MySQL-ähnlichen Abfragen. Dies gibt einem technischen Datenbankadministrator mehr Möglichkeiten bei der Fehlersuche und -korrektur. Neben den einfachen Abfragen bietet MongoDB Map/Reduce-Funktionalitäten, um schnell große Datenmengen auszuwerten zu können. [2]

GlassFish v. 3.1.1 ist die Referenzimplementierung von Oracle zum JavaEE-Standard und im Gegensatz zur Konkurrenz wie JBoss leichtgewichtig und durch die Webschnittstelle zur Konfiguration gut handhabbar. In Folge dessen ist dieser Applikationsserver sehr performant. Durch das Persistence Framework TopLink, welches auch von Oracle entwickelt wird, bietet dieser Applikationsserver eine gute Konnektivität zu den verwendeten Datenbankprodukten.

5.6. Zusammenfassung der Serverfunktionalität

Der lange verfolgte und favorisierte Architekturansatz „Thin-Client und Server“ stellte sich nach dreimonatiger Entwicklungsphase des Servers als nicht nutzbar heraus. Die Aufteilung der Server-Software in mehrere Module und die eigenverantwortliche Implementierung der Teile durch die Teammitglieder funktionierte gut, so dass alle Module mit einem geringen Verzug von drei Wochen fertig gestellt wurden. Als es um die Kopplung der Module ging wurde ersichtlich, dass die Bibliothek zur Verwaltung der Verbindungen zu den einzelnen Ressourcen für eine Clientanwendung funktional war, allerdings nicht in einer verteilten Umgebung eingesetzt werden konnte. Ein Umschreiben dieser Bibliothek kam auf Grund der knappen Zeit nicht in Frage, so dass beschlossen wurde, die Bibliothek zur Nutzung an den Clientgeräten zu optimieren und den Server von der Funktion des Datentransfers zu entbinden. Somit wurden Module wie die Login-Verwaltung eines jeden Benutzers zu einer entfernten Ressource überflüssig und auch der Dienst für den Datentransfer wurde ohne passende Bibliothek obsolet.

Der implementierte Ansatz der SVDI ist ein Fat-Client und ein Server zur Verwaltung der Metadaten, der Benutzer und der Ressourcen, um dem Benutzer am Client einen einheitlichen Informationsbestand für seine Daten und die Ressourcen, auf die er berechtigt ist zu zugreifen, zu bieten. Zu einer jeden Ressource existiert nun ein separates Feld, dass ein byte-Array speichern kann und eindeutig einem Benutzer und einer Ressource zugeordnet ist. Das Feld kann vom Client beliebig beschrieben werden und symbolisiert, dass ein Client diese Ressource regelmäßig für seine Arbeit nutzt. Wird der Datensatz <userid, ressourcenid, Userinfo> gelöscht, bedeutet das, dass ein Benutzer diese Ressource für sich selbst entfernt hat. Die Ressource existiert für die Relation zu den anderen Benutzern und den Metadaten weiter.

[2] `http://www.mongodb.org/display/DOCS/Comparing+Mongo+DB+and+Couch+DB`

So wanderten die Funktionalitäten für die Login-Verwaltung in den Client und auch die Dateitransfers zwischen dem User und den entfernten Ressourcen werden nun vom Client übernommen. Um nun das Problem mit Inkonsistenzen zu lösen, die auftreten können, wenn zwei User an der selben Datei gearbeitet haben und diese nun hochladen wollen, unterstützt der Server den Client beim Auflösen des Konflikts. So kann der schnellere Benutzer mit seinem Client, das Metadatum als gesperrt markieren, um jedem anderen zu signalisieren, das der Datensatz oder die Datensätze gerade auf der entfernten Ressource verändert werden. Des weiteren bietet der Server dem Client die Möglichkeit zu jedem Metadatum beliebige Informationen in Form von Schlüssel=Wert hinzuzufügen, die jeder andere Benutzer lesen, aber nicht löschen oder ändern kann. So kann ein Benutzer die Änderungen an einem Datensatz nachvollziehen, wenn die anderen Benutzer Informationen hinzufügen.

5.7. Client

Der zweite Bestandteil der SVDI ist der Client, über den der Benutzer seine Eingaben tätigt. Der Client selbst, ist dabei für den Dateitransfer von und zu entfernten Ressourcen zuständig. Der Datentransfer erfolgt damit direkt vom Client zur Ressource, ohne einen weiteren Umweg über den SVDI-Server nehmen zu müssen. Außer dass über diese Lösung das weiter oben beschriebene Problem der Verbindungsverwaltung innerhalb der genutzten Bibliothek gelöst wurde, bietet eine direkt Verbindung zwischen Client und Ressource weiterhin den Vorteil, dass der SVDI-Server nicht ein möglicher Flaschenhals innerhalb der Verbindungskette werden kann. Darüber hinaus synchronisiert der Client bei Bedarf die Metadaten mit dem SVDI-Server, so dass dem Benutzer möglichst immer die aktuellste Version der Metadaten zur Verfügung stehen. Die Kommunikation mit dem SVDI-Server erfolgt per SOAP-Nachricht und wird verschlüsselt über das HTTPS-Protokoll übertragen. Eine verschlüsselte Verbindung wurde innerhalb der Projektgruppe an dieser Stelle als notwendig angesehen, so dass die Daten und Metadaten der Forscher aus der Verbindung heraus nicht einfach ausgelesen werden können. Das HTTPS bietet hierfür entsprechenden Schutz. Eine detaillierte Beschreibung der Funktionalitäten findet sich in Anhang **??**.

5.8. Metadaten

Die gespeicherten Metadaten lassen sich in zwei Kategorien unterteilen: Die erste Kategorie sind die Systemmetadaten. Dazu gehören die Metadaten, die eine feste Struktur besitzen und für jede Datei vorhanden sind. Zum Beispiel gehören der Dateiname und der Speicherort, welche es ermöglichen eine Datei eindeutig zu identifizieren, zu den System-

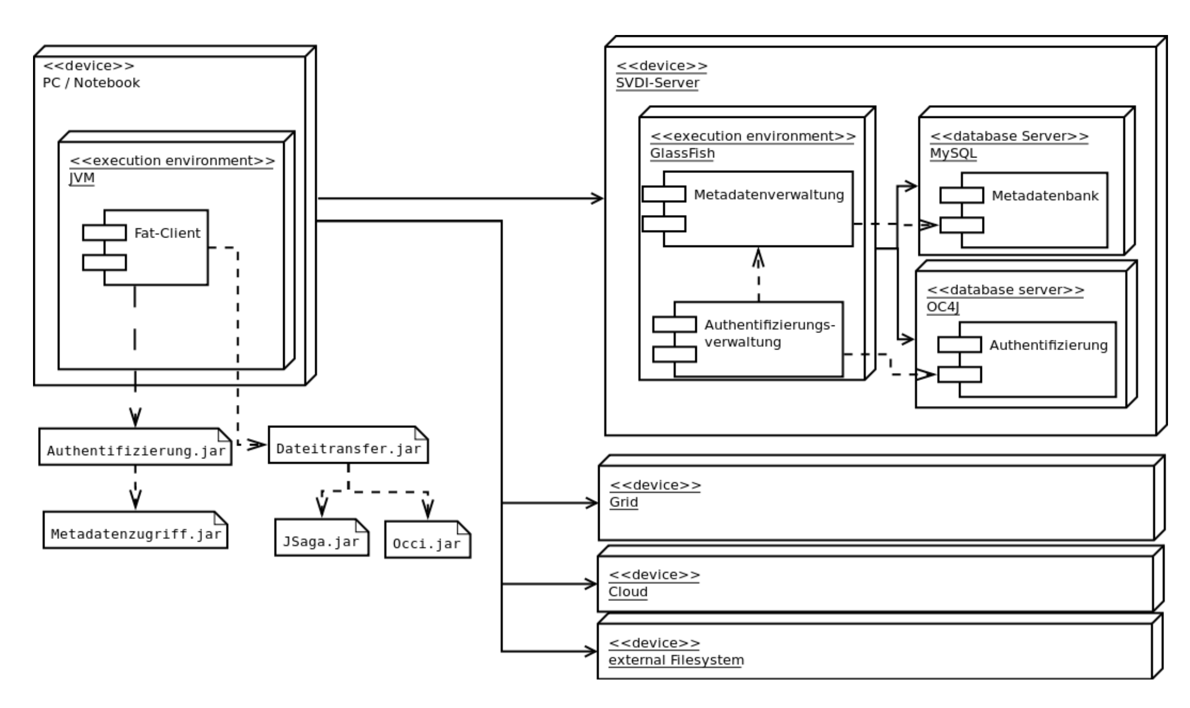


Abbildung 5.1.: Darstellung des Fat-Client-Ansatzes als Verteilungsdiagramm

metadaten. Die andere Kategorie sind Benutzermetadaten. Dem Benutzer und damit insbesondere auch der Clientanwendung wird ermöglicht beliebige Metadaten in Form von Key-Value-Paaren hinzuzufügen. Durch die Verwendung von MongoDB ist es ohne großen Mehraufwand möglich beide Arten von Metadaten zusammen in einem Objekt zu speichern. Zudem ist die Suche nach Metadaten über die Benutzermetadaten so wie über die Systemmetadaten möglich. Eine komplette Übersicht der gespeicherten Metadaten ist in Tabelle 5.1 zu sehen.

Id	Interne ID des Metadatums. Bestimmt ob beim Setzen eines Metadatums ein vorhandenes geändert oder ein neues erzeugt werden soll.
Name	Absoluter Pfad der Datei auf der Ressource
ResourceId	ID der Ressource auf der die Datei liegt
Size	Dateigröße
DateCreated	Erzeugungszeitpunkt des Metadatums
Categories	Kategorien zu denen die Datei gehört (Tagging)
Locked	Zeigt an, ob die Datei für die Änderung durch andere Benutzer gesperrt ist
custommetadata	Key-Value-Paare für Benutzermetadaten (pro Benutzer)
lastchanged	Letztes Änderungsdatum (pro Benutzer)

Tabelle 5.1.: Übersicht der gespeicherten Metadaten

Die Metadaten werden dabei sowohl auf der ursprünglichen Ressource gespeichert (sofern es sich um Systemmetadaten handelt) als auch im Client. Die im Client gespeicherten Benutzermetadaten müssen dabei mit dem SVDI-Server synchronisiert werden, was direkt bei einer Änderung durch den Benutzer erfolgt. Die Metadaten werden im Key-Value Format in einem Objekt verpackt und anschließend an den SVDI-Server übertragen. Die Suche in den Metadaten erfolgt ähnlich. Sobald der Benutzer die entsprechenden Suchbegriffe in die Suchmaske eingegeben hat, werden diese per Webservice-Aufruf an den SVDI-Server übertragen, der eine vollständige Suche auf allen ihm bekannten Daten in der Metadaten-Datenbank durchführt. Im Falle eines Fehlers bei der Kommunikation zwischen Client und Server, wird der Benutzer darüber mit einer Fehlermeldung informiert, ansonsten wird er auf die erfolgreiche Aktualisierung der Metadaten hingewiesen bzw. bekommt eine Liste der gefundenen Suchergebnisse, die allerdings auch leer sein kann.

6. Implementierung

Dieses Kapitel beschäftigt sich mit der praktischen Implementierung von Client und Server. Der Fokus liegt dabei auf der Aufteilung der Funktionalität auf die einzelnen Klassen, deren Zuordnung zu Paketen beziehungsweise Komponenten sowie die Beschreibung der typischen Aufgaben einzelner Klassen und der Interaktion der Klassen untereinander.

6.1. Client

Für die Implementierung des Clients hat sich das Client-Team am bekannten MVC-Entwicklungsmuster orientiert. Der Client gliedert sich also primär in die drei Pakete „Model“, „View“ und „Controller“. Des Weiteren wurden mit dem Start- und dem Proxy-Paket zwei weitere Pakete erstellt. Das Start-Paket enthält dabei die für unterschiedliche Startkonfigurationen benötigten Klassen, das Proxy-Paket realisiert die Schnittstellen des Clients mit dem SVDI-Server sowie den Datenservern (FTP und GRID).

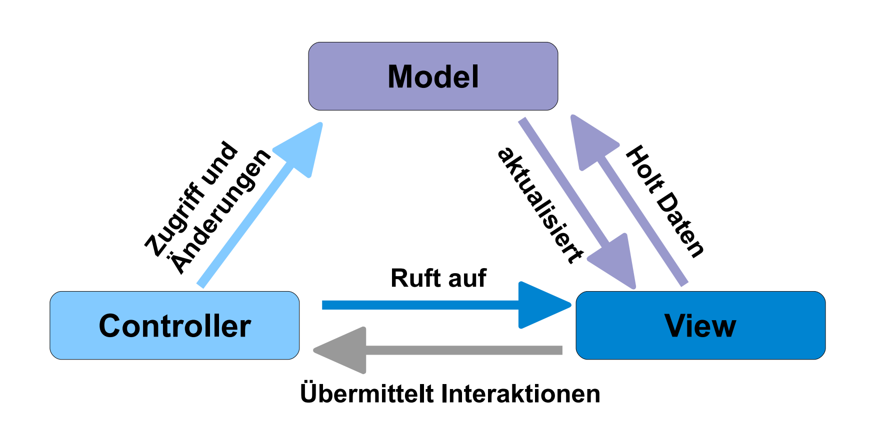


Abbildung 6.1.: Das Model-View-Controller-Konzept

6.1.1. Modell

Das Modell definiert die grundlegende Struktur des Clients, im gleichnamigen Paket befinden sich daher alle für die Datenhaltung benötigten Klassen. Für die Verwaltung und Identifikation der Benutzer wird die Klasse **UsernamePasswordToken** genutzt, die Informationen wie den Benutzernamen, die E-Mail-Adresse sowie Session-Informationen speichert. Verknüpft mit einer Instanz dieser Klasse ist eine Instanz von **UserCache**, in der die vom jeweiligen Benutzer gebrauchten Ressourcen, Projekte und Zertifikate in einer dynamischen Datenstruktur gespeichert werden.
Die Klasse **Ressource** dient als gemeinsame Oberklasse für alle speziellen Ressourcentypen, von denen bisher die Klassen **SFTPRessource**, **FTPRessource** und **GridRessource** implementiert sind. Diese Unterklassen enthalten eine, jeweils auf die entfernte Ressource angepasste **connect()**-Methode. Dies ist erforderlich, da diese Ressourcen keine einheitlichen Schnittstellen nach außen bieten und so beispielsweise bei FTP-Ressourcen die Benutzeridentifikation über Benutzername und Kennwort erfolgt, während bei Gridressourcen ein Proxyzertifikat verwendet wird, das allerdings erst noch aus dem hinterlegten Userzertifikat generiert werden muss.
In der Klasse **Projekt**, die für die Verwaltung der vom Benutzer angelegten Projekte zuständig sin, werden primär Instanzen der Klasse **FileHeader** gespeichert. **FileHeader** sind dabei Verweise auf Dateien, sie beinhalten Name, Metadaten, die Ressource, auf der die Datei gespeichert ist sowie deren lokalen Pfad auf dieser Ressource. Die Benutzerzertifikate schließlich werden in Instanzen der Klasse **Certificate** gespeichert und enthalten ausser dem Zertifikat selbst noch eine Pfadangabe zum Original-Zertifikat.

6.1.2. Controller

Der Controller gliedert sich in die drei Klassen **FileController**, **MetadatenController** und **Controller**. In der Klasse **FileController** wurden alle Methoden implementiert, die für lokale und externe Datenmanipulation, wie das Löschen und Speichern einer lokalen Datei, Löschen und Speichern einer Datei auf einer entfernten Ressource, etc. notwendig sind. Auch der Datentransfer, also das Hoch- beziehungsweise Herunterladen von Dateien, erfolgt über den **FileController**.
Der **MetadatenControler** ist für die Verwaltung der Metadaten zuständig. Dies beinhaltet insbesondere das Hinzufügen, Löschen und Ändern der Metadaten sowie deren Synchronisation mit dem SVDI-Server. Dabei macht es keinen Unterschied, ob die Metadaten sich auf lokale oder entfernte Dateien beziehen, in beiden Fällen sind Methoden dieser Klasse dafür zuständig.
Die Klasse **Controller** schließlich ist eine Sammelklasse, die alle anderen Methoden zur Steuerung des Clients beinhaltet. So befinden sich hier Methoden zur Erstellung und zum Login von Benutzern, zur Verwaltung von Benutzerinformationen sowie zur Manipulation von Projekten, Ressourcen, und Zertifikaten. Insbesondere ist die Klasse **Controller**

für die Erstellung von Proxyzertifikaten sowie die Verschlüsselung von Passwörtern durch einen MD5-Hashing-Algorithmus zuständig.
Alle drei Controller-Klassen werden gemeinsam von einer Instanz der Klasse **InitController** gekapselt. Diese initialisiert bei Programmstart die drei Controller und bildet eine gemeinsame Schnittstelle für alle drei Klasen zur View-Schicht.

6.1.3. View

Die für die Benutzeroberfläche erforderlichen GUI-Klassen des Clients befinden sich im View-Paket und basieren auf der Bibliothek „Swing". Beim Programmstart wird eine Instanz der Klasse **LoginWindow** generiert, welche ein Formular zur Verfügung stellt, über das sich der Benutzer mittels E-Mail und Passwort beim SVDI-Server identifizieren kann.

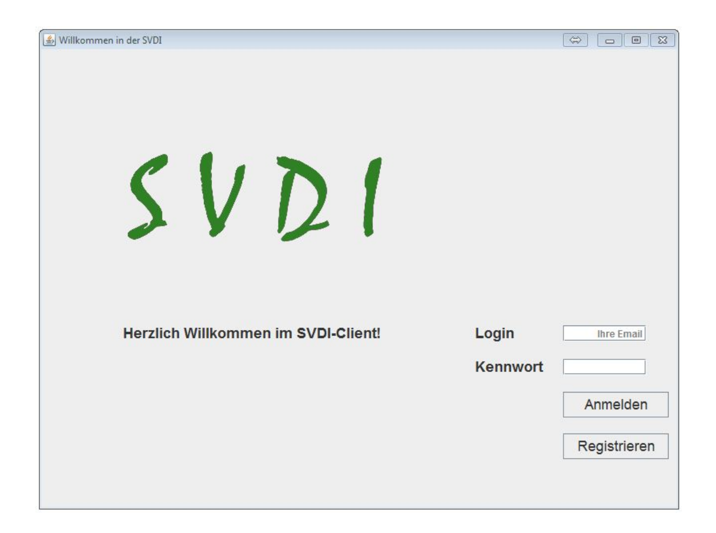


Abbildung 6.2.: Login-Fenster des SVDI-Clients

Außerdem besteht hier die Möglichkeit, ein Objekt der Klasse **RegistrationWindows** zu erzeugen, mit dem ein neues Benutzerkonto registriert werden kann. Nach Eingabe der erforderlichen Daten in das Registrierungsformular und der erfolgreichen Erstellung eines Benutzerkontos kann sich der Benutzer anschließend über das **LoginWindow** einloggen. Um dem Endbenutzer die Orientierung in der SVDI sowie die Handhabung des Client-Programms zu vereinfachen, wurden für typische Aufgabenstellungen Schritt-für-Schritt-Anleitungen in Form von Wizards implementiert. Nach der erfolgreichen Anmeldung beim SVDI-Server wird der **StartWizzard** ausgeführt, über den häufig genutzte Funktionalitäten, wie das Hinzufügen lokaler Dateien, das Anlegen eines Projekts, das Erstellen einer Verbindung zu einer entfernten Ressource und die Suchfunktion direkt angesprochen werden können. Sollte keine dieser Funktionen gewählt werden, wird das

MainWindow angezeigt, das sich in zwei Segmente gliedert, die Projektansicht (links) sowie das Informationsfenster (rechts).

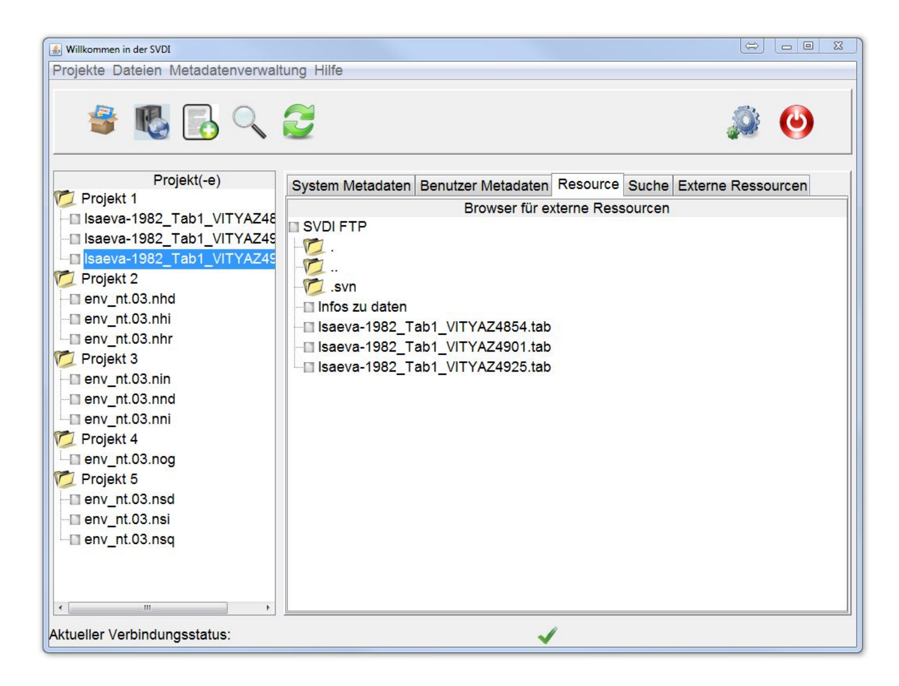


Abbildung 6.3.: Hauptfenster des SVDI-Clients

In der Projektansicht (darstellt in Abb. 6.4) befindet sich eine durch **JTree** implementierte Baumstruktur, welche die den Projekten zugeordneten Dateien enthält. Dateien können dabei in der Projektansicht mittels „Drag and Drop“ verschoben werden. Das Informationsfenster in der rechten Bildschirmhälfte bietet eine Vielzahl von Informationen bezüglich der system- und nutzerdefinierten Metadaten sowie des Inhalts einer Ressource. Außerdem befindet sich hier eine Auflistung der zum aktuellen Benutzer registrierten Ressourcen und ein Suchfenster. Diese Informationen bzw. Funktionalitäten sind dabei mittels **JTabbedPane** in einzelnen, inhaltlich zusammenpassenden Tabs bzw. Reitern untergebracht. Wie für Fensteranwendungen üblich, verfügt auch der SVDI-Client über eine Menüleiste, über die sich alle Funktionalitäten anwählen lassen, sowie über eine Toolbar, die es erlaubt, bestimmte Funktionen oder Wizards mittels eines Icons auszuführen. Am unteren Bildschirmrand schließlich befindet sich die Statusleiste, die dem Benutzer anzeigt, mit welchen entfernten Ressourcen er gerade verbunden ist, und welchen Status die jeweiligen Verbindungen aufweisen.

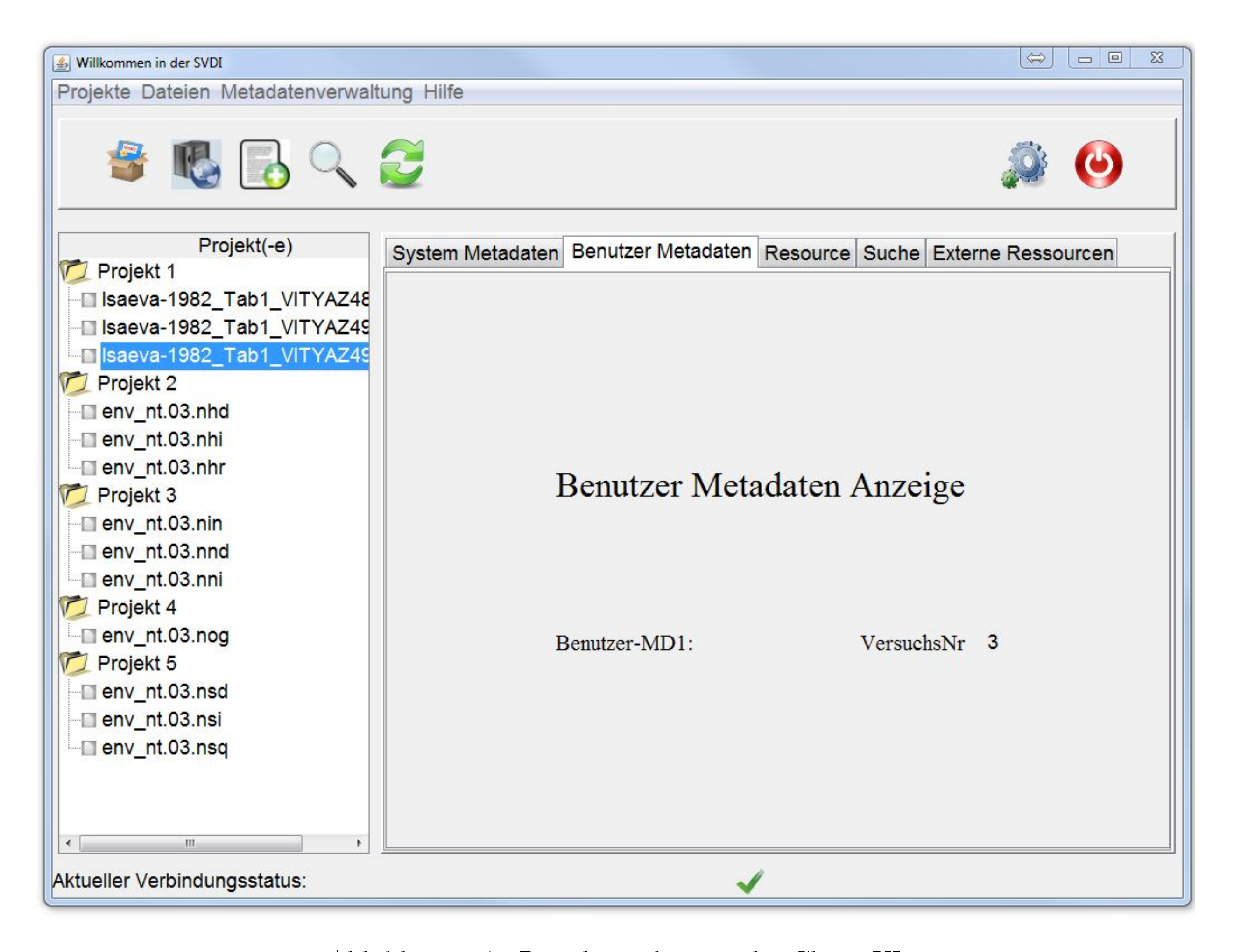


Abbildung 6.4.: Projektstruktur in der Client-UI

6.1.4. Funktionalität

Der SVDI-Client dient als Benutzer-Schnittstelle zur SVDI und stellt eine Reihe wichtiger Funktionen zur Verfügung. Die Organisation der Dateien in der SVDI erfolgt projektbasiert. Der Benutzer kann beliebig viele Projekte mit beliebig vielen Dateien erstellen. Die einzige bestehende Einschränkung ist, dass zwei Projekte nicht den gleichen Namen haben dürfen. Gleiches gilt für zwei Dateien innerhalb des gleichen Projekts. Sollte der Benutzer dennoch versuchen, zwei Projekten oder Dateien den gleichen Namen zu geben, so wird er aufgefordert, eines der Projekte bzw. eine der Dateien umzubenennen. Die Projektstruktur ermöglicht dem Benutzer einen schnellen, direkten Zugriff auf für ihn wichtige Dateien, da sie auf die lokale Festplatte des Benutzers abgebildet werden und er sich nicht mehr durch Ordnerhierarchien auf entfernten Ressourcen hangeln muss. Für jedes Projekt liegt im Verzeichnis „User_Home/SVDI_HOME/SVDI_Downloads“ ein namensgleicher Ordner, in dem die vom Client heruntergeladenen Dateien gespeichert

werden.

Die zweite wichtige Funktionalität des Clients liegt in seiner Möglichkeit, sich mit entfernten Datenquellen zu verbinden, dort zu navigieren und Dateien herauf- sowie herunterzuladen. Zurzeit unterstützt der Client die Verbindung zu FTP- und SFTP-Servern sowie zu diversen Gridmiddlewares. Wird eine entfernte Ressource mit dem Client besucht, indem deren Pfad und die zugehörigen Authentifizierungsdaten sowie optional ein Name eingegeben wird, so werden die auf der Ressource liegenden Ordner und Dateien im Ressourcenbrowser des Clients angezeigt, mittels dessen die Navigation innerhalb der Ressource erfolgt. Hierbei werden natürlich nur die System-Metadaten wie Name, Pfad etc. von der Ressource heruntergeladen und nicht die vollständigen Dateien, um die Navigation nicht unnötig zu verlangsamen. Die Manipulation der Dateien erfolgt ebenfalls durch den Ressourcen-Browser, dieser unterstützt das Löschen, Umbenennen sowie Herauf- und Herunterladen der Dateien. Außerdem können Dateien über den Ressourcen-Browser einem Projekt hinzugefügt werden.

Das dritte Merkmal des SVDI-Clients ist die Möglichkeit zum Annotieren der Daten, die sich innerhalb von Projekten befinden. So kann der Benutzer über entsprechende Eingabefelder eigene Metadaten zu Dateien hinzufügen und über eine Suchfunktion Dateien nach diesen Metadaten filtern.

6.1.5. Schwierigkeiten bei der Implementierung

Das erste Hindernis bei der Entwicklung des Clients waren die teils unklaren Anforderungen an diesen zu Beginn der Implementierungsphase. Da die Architektur und somit die Verteilung der Funktionalität zwischen Server und Client mehrmals geändert wurde, gab es hier einige Unklarheiten, die jedoch durch intensivere Kommunikation ausgeräumt werden konnten. Insbesondere die regelmäßigen Treffen zwischen Client- und Serverteam waren hier hilfreich. Programmiertechnisch lag das größte Problem in der mangelnden Praxis-Erfahrung der Teilnehmer. So hatte kaum ein Teilnehmer zuvor eine verteilte Anwendung programmiert, was zu Problemen beim Softwaredesign führte. Außerdem fehlten teilweise Detailkenntnisse einzelner Swing-Klassen (hier sei insbesondere JTree zu nennen), die zu einer längeren Einarbeitungszeit und Experimentierphase und somit schlussendlich zu Verzögerungen in der Implementierung führten. Auch bei der Verwendung von externen Bibliotheken und Frameworks kam es zu Problemen. So war die Einarbeitungszeit und der Anpassungsaufwand bisweilen recht hoch, auch wenn durch die Verwendung der Bibliotheken viel Zeit im Vergleich zu einer vollständigen Neuentwicklung eingespart wurde. Außerdem kam es immer wieder zu kleineren Problemen, von denen hier einige exemplarisch genannt werden sollen.

Problem mit Serialisierung von UserCache Beim User-Cache handelt es sich um Ressourcen- und Projektinformationen, die in einer Datei lokal gespeichert werden, damit diese auch in der nächsten Programmsitzung verfügbar sind. Zunächst wurde im Client ein klassisches MVC-Muster genutzt. Es wurde also zu jedem Ressourcentyp eine entsprechende Klasse im Modell hinzugefügt. Die Funktionalitäten dieser Klassen sollten ebenfalls in Modell-Klassen liegen, nutzten jedoch JSAGA-Methoden, die nicht serialisierbar waren. **Lösung:** Die Ressourcenklassen wurden in eine Datenhaltungs- und eine Controllerklasse aufgeteilt, so dass die interessanten Ressourcendaten serialisiert werden konnten.

Problem bei der Anzeige von verschiedenen Dateitypen In der GUI sollten alle mögliche Dateien anzeigt werden können, unabhängig davon, ob es sich um lokale oder entfernte Dateien handelt und ob diese über benutzerdefinierte Metadaten verfügen oder nicht. Dabei war es jedoch möglich, dass sich diese Dateitypen änderten (bei Up- oder Download, Änderung der Metadaten, etc.), so dass der Zustand einer Datei an geeigneter Stelle gespeichert werden musste. **Lösung:** Es wurde eine universelle Klasse FileHeader hinzugefügt, die alle möglichen Informationen über Dateien und Metadaten enthält. So wurde immer die Liste vom Typ FileHeader in der GUI angezeigt, egal ob es sich um eine entfernte oder lokale Datei handelte.

Probleme bei der Modellumwandlung Die Modellklassen im Client unterschieden sich von den DTO-Klassen, die zum Server geschickt werden sollten. **Lösung:** Es wurden generische Klassen erstellt, die Clientklassen in DTO-Klassen umwandeln und umgekehrt.

6.2. Server

Nach dem gewählten Ansatz, der in Kapitel 5 beschrieben wurde, befindet sich die Verwaltung, der für den Datentransfer relevanten Informationen, im SVDI-Server. Dieser ist ebenfalls die Kontrollinstanz, die die Zugriffe auf die bekannten Daten kontrolliert. Um diese Funktionalität zu gewährleisten ist der Server logisch in die Module Benutzerverwaltung, Metadatenverwaltung, Dateitransferservice, Logininformationsverwaltung und der Ressourcenverwaltung aufgeteilt. Die Informationen aus den Modulen erhält der SVDI-Client über die Webservice-Schnittstelle, die derzeit in SOAP implementiert ist.

Auf Grund von Problemen bei der Umsetzung des Dateitransfers vom Server zu den einzelnen Ressourcen im Namen eines jeden angemeldeten Benutzers, entschied sich die Gruppe allerdings für den Fat-Client-Ansatz (beschrieben im Kapitel 5). Dies lässt einige Module, die im Folgenden beschrieben sind überflüssig werden und ändert die Struktur des Servers.

6.2.1. Benutzerverwaltung

Die Benutzerverwaltung ermöglicht das Hinterlegen von Benutzerinformationen für die SVDI. Sie stellt die Funktionalität für die Registrierung und den Login-Vorgang sowie die Änderung der Benutzerinformationen zur Verfügung. Des Weiteren findet sich in diesem Modul die Sessionverwaltung wieder, mittels derer es den anderen Modulen möglich ist den Zustand des Benutzers innerhalb des Systems zu prüfen.

6.2.2. Ressourcenverwaltung

Die Ressourcenverwaltung organisiert die Informationen der Speicherorte, die ein Benutzer im System hinterlegt. Existiert eine Ressource noch nicht, kann diese angelegt werden. Ein Benutzer bekommt diese Ressource zugewiesen, wenn er für diese Login-Informationen bzw. eine UserInfo hinterlegt. Ressourcen werden in diesem Modul nur ein einziges Mal gespeichert und können nicht gelöscht werden, um Inkonsistenzen zu verhindern. Das Löschen einer Ressource stellt sich für einen Benutzer so dar, dass er die UserInfo zu einer Ressource entfernt, so dass diese in seiner Ressourcenliste im Client nicht mehr aufgeführt wird. So ist die Ressource für alle anderen Benutzer und die hinterlegten Metadaten noch immer aktuell und sichtbar.

6.2.3. Logininformationsverwaltung

Die Logininformationsverwaltung hat keine eigene Schnittstelle nach außen, stattdessen stellt sie ihre Funktionen dem Benutzer indirekt über die Ressourcenverwaltung und die Metadataverwaltung zur Verfügung. Eine Logininformation besteht aus dem Tripel Benutzer, Ressource und eigentliche Logindaten. Der Benutzer stellt hierbei den Besitzer der Logininformation dar, die Ressource ist der Server o.ä. auf den mittels der Logininformationen zugegriffen werden kann und die Logindaten sind schließlich die Daten die für den jeweiligen Zugriff nötig sind, wie zum Beispiel Benutzername und Passwort oder ein Zertifikat.

6.2.4. UserInfoverwaltung

Diese UserInfoverwaltung ist gegen Ende des Projektes im Server implementiert worden und ersetzt die Logininformationsverwaltung. Diese Funktionalität steht der Client-Anwendung über die Webservice-Schnittstelle der Ressourcenverwaltung zur Verfügung. In der Datenbank existiert eine Tabelle, die zur Verwaltung der UserInfos zur Verfügung

steht. Wie bei der LoginInformationsverwaltung hat ein Benutzer pro Ressource die Möglichkeit eine Information zu hinterlegen. Diese Information liegt als Byte-Array vor und kann ein serialisiertes Objekt enthalten. Diese Information wird vom Server nur entgegengenommen und verwahrt, aber nicht verstanden. Es findet serverseitig keine Prüfung des Inhaltes statt.

6.2.5. Aufgabe des Servers

Der Server ist ein reiner Informationsspeicher zur Verwaltung der Ressourcen und Metadaten. Er ermöglicht dem Client spezifische Informationen eines Benutzers zentral zu hinterlegen, verwaltet die Informationen, die für alle Benutzer interessant sind und kontrolliert den Zugriff auf die hinterlegten Metadaten- und Ressourceninformationen. Er erlaubt einem User den Client auf verschiedenen Endgeräten auszuführen und dennoch immer die gleiche Sicht auf die Daten zu haben.

6.2.6. Entwicklung des Servers

Die Module des Servers wurden von einem vier Mann starken Team bearbeitet. Jedes Mitglied des Teams bekam einen Verantwortungsbereich und ein Arbeitspaket, dass selbst zu spezifizieren, abzuschätzen und umzusetzen war, zugewiesen. Lediglich der grundlegende Ablauf der Kommunikation wurde im Vorfeld mittels UML-Aktivitätsdiagrammen gemeinsam spezifiziert. Die Entwicklung erfolgte parallel zur Entwicklung des Clients, was einen hohen Grad an Zusammenarbeit und Kommunikation erforderte. Dringende Fragen und Diskussionen wurden in einem wöchentlichen Treffen abseits der allgemeinen PG-Sitzung besprochen und diskutiert. Durch die klare Abgrenzung, der Fähigkeiten des Clients und des Servers genügte es die Web Service-Schnittstellen zu definieren und dem Client mitzuteilen, was er bei Erfolg und Fehlschlag zu erwarten hatte. Diese Informationen wurden dem Client-Team als Use-Case Beschreibung und in Form eines Schnittstellendokuments angeboten.

Softwarearchitektur

Drei Schichten mit unterschiedlichen Aufgaben definieren die Serversoftware, vgl. Abb. 6.5. Die Definition der Schichten dient der besseren Wartbarkeit und orientiert sich am MVC-Pattern. Zusätzlich wurde beim Design darauf geachtet, weitere bekannte Software-Pattern[GHJV95] wie „Adapter“ und „Singleton“ zu verwenden, um einen hohen Grad an Wartbarkeit und Erweiterbarkeit zu gewährleisten.

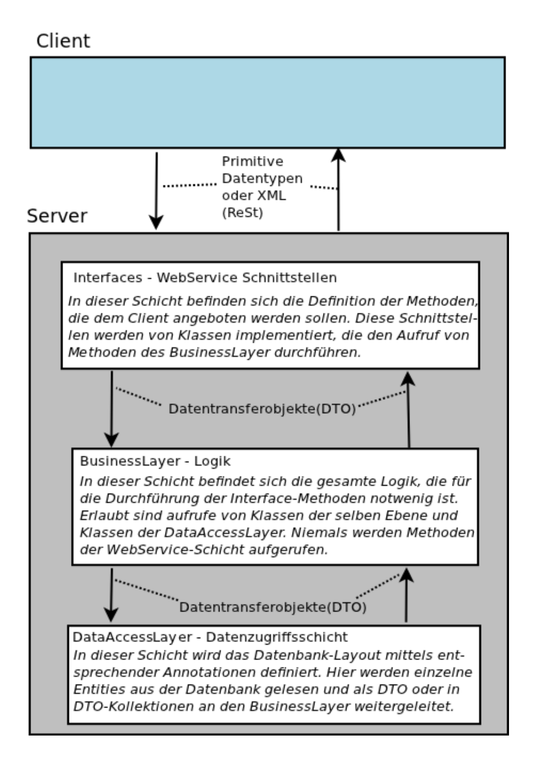


Abbildung 6.5.: Serverarchitektur

Die Schichten des Servers

Die oberste Sicht, der **Web Service Layer**, bietet dem Client die erforderliche Schnittstelle zur Dateneingabe und -gewinnung. Die Methodenaufrufe der Web-Service-Schnittstelle sind so definiert, dass die Abarbeitungsreihenfolge und -kontrolle dem Server obliegt. So gliedert sich ein grobgranularer Aufruf des Clients im Server-Code, in mehrere feingranulare Aufrufe, die für eine korrekte Verarbeitung oder eine aussagekräftige Fehlermeldung sorgen. Es wurde versucht die Schnittstelle so zu entwerfen, dass die Clientsoftware zur Durchführung einer Aktion, zur Informationsabfrage oder -eingabe, nie mehr als einen Aufruf tätigen muss. Dem Client werden die Softwareschnittstellen über ein WSDL-Dokument angeboten, mit denen dieser den für einen Aufruf benötigten Quellcode generieren kann.

Unter dem Web Service Layer ist der **Business Layer** angesiedelt, der die Serverlogik enthält. Diese Schicht kontrolliert die Abarbeitungsreihenfolge zur Erstellung eines, aus den Clientaufrufen resultierenden, Ergebnisses. Diese Schicht hat keinen Zugriff auf die eigene oder eine Webservice-Schnittstelle eines anderen Moduls. Assoziationen zu anderen Modulen werden auf dieser Softwareebene definiert und genutzt, wenn Informationen aus anderen Modulen benötigt werden. Innerhalb des Business Layers wird über DTOs kommuniziert, da diese, über die darüber liegende Web Service Schnittstelle, an den

Client geliefert werden.

Die unterste Schicht der Softwarearchitektur ist der **Data Access Layer (DAL)**. In dieser Schicht finden die Datenbankzugriffe statt und nur in dieser Schicht werden die Datenbank-Entitäten manipuliert. Der Zugriff auf die Datenbank wird mittels Java Persistance API (JPA) realisiert. Für jedes Modul existiert eine Serviceklasse, die Datenbankabfragen generiert und das Ergebnis einer Abfrage an den Business Layer sendet, der dieses Ergebnis weiterverarbeitet. Dieser Layer erstellt keine eigenen Exceptions, sondern leitet die Datenbankfehler lediglich an den Business Layer weiter. So dient diese Schicht nur der Datenspeicherung und -gewinnung.

Konzept der Ausnahmebehandlung

Die Ausnahmebehandlung ist bei Serverarchitekturen sehr wichtig. Einerseits müssen Fehlermeldungen für den Client und den bedienenden Benutzer aussagekräftig und hilfreich sein, andererseits dürfen diese Meldungen nicht zu viel von der Architektur verraten und keinen zu großen Detaillierungsgrad besitzen. Das Exception-Handling wird hauptsächlich im Business Layer und in den Validatoren verwendet. Im Business Layer werden die Fehlermeldungen der Datenbank in selbst definierte Exceptions umgewandelt, um dem Client den Ursprung des Fehlers zu verbergen, so dass diese nicht als mögliche Angriffspunkte gegen den Server genutzt werden können. Zur Meldung von Fehlern existieren die folgenden Exception-Typen:

FaultBean Die FaultBean-Exception wird geworfen, wenn ein allgemeiner Fehler auftritt.

InvalidInput Diese Exception wird geworfen, wenn vom Client falsche Eingaben übergeben werden, wie beispielsweise eine ungültige Session-ID. Diese Exception wird am häufigsten geworfen.

MetadataTooOld Eine MetadataTooOld-Exception wird geworfen, wenn ein anderer Client den Datensatz in der Datenbank bereits geändert hat und ein veralteter Datensatz geändert werden soll.

NoConnectionToResource Diese Art von Exception ist veraltet und wurde ursprünglich vom FileTransferService geworfen. Da der Dateitransfer nun allerdings über den Client direkt durchgeführt wird, wird diese Exception aktuell nicht geworfen.

NotExpectedException Diese Exception diente zur Kapselung der JSaga-Exception NotImplementedException, die geworfen wurde, wenn ein Protokoll verwendet werden sollte, dass nicht implementiert ist. Da allerdings der Client vom Server vorgegeben bekam, welche Ressourcen angesprochen werden konnten, sollte diese Exception nie gewor-

fen werden. Aktuell dient diese Fehlermeldung dazu Ausnahmen zu kapseln, die von einer API geworfen werden und von der SVDI nicht sinnvoll behandelt werden können und von temporärer Natur sind.

NotSufficientRights Diese Exception meldet dem Client und dem Benutzer, dass der aktuelle Benutzer nicht die Berechtigungen hat eine Aktion, die von Server kontrolliert wird auszuführen.

6.2.7. Server-Entwicklung

Nach der Spezifikationsphase, in der zahlreiche UML-Diagramme zur Entwicklungsgrundlage entstanden, wurden in der ersten Stufe die Schnittstellen für den User-Login und die Metadatenspeicherung erarbeitet. Zur Definition der Schnittstellen wurde das Client-Team zu separaten Treffen des Server-Teams eingeladen. In diesem wurden in mehreren Gruppen die Methoden der Web-Service-Schnittstelle definiert. So erhielt das Client-Team einen Einblick in die Serverentwicklung und konnte die Funktionalitäten der Methoden beeinflussen. Nach der Definitionsphase wurden die gesammelten Methoden auf die verschiedenen Server-Module verteilt und in kleineren Teams die Funktionalität ausgearbeitet.

Es war angedacht die Serverentwicklung parallel zur Entwicklung des Clients durchzuführen, so dass es möglich ist Schnittstellen parallel zur GUI-Entwicklung zu testen und Probleme schnell zu beheben, bevor an weiterer Funktionalität gearbeitet wurde. Die ersten Module, die implementiert wurden waren die Userverwaltung und die Metadatenverwaltung. Erst nach deren Fertigstellung wurde die Implementierung der Ressourcenverwaltung und des Datentransfers in Angriff genommen.

Die Rolle der Data Transfer Objects (DTO)
Eine weitere Absprache der Teams betraf, die Definition von Transferobjekten. Jedes Datenbankobjekt, welches der Client benötigt, wurde zusätzlich als DataTransferObjekt (DTO) abgebildet. Diese Objekte enthalten die einfachen Daten der Datenbankobjekte. Zur Definition der Objektabhängigkeiten enthält jedes DTO ein Feld für die Datenbank-ID und die Fremdschlüssel zu den in Beziehung stehenden Objekten. Der Vorteil dieser Objekte liegt darin, dass nicht jede Methodensignatur angepasst werden muss, wenn sich an den Objekten etwas ändert. Das macht Änderungen schneller und flexibler. Innerhalb des Servers wird direkt mit den Datenbankobjekten gearbeitet. Nur zur Kommunikation über die Servergrenzen hinaus, werden die Objekte für den Transfer umgewandelt. Ein an den Server übergebenes DTO muss nur die Felder enthalten, die für den Aufruf der Methode erforderlich sind. So ist es nicht erforderlich alle Felder eines DTOs auszufüllen, um vom Server eine korrekte Antwort zu erhalten. So könnten beispielsweise bei der Registrierung Felder wie Session-ID, User-ID und Login-Timestamp leer bleiben, da der Client diese erst nach einem erfolgreichen Login erhält.

7. Tools und Bibliotheken

Dieses Kapitel beschäftigt sich mit den von der Projektgruppe zur Entwicklung genutzten Tools und Bibliotheken. Eigenständige Softwarekomponenten, wie Webserver oder Datenbanken werden hier nicht beschrieben.

7.1. Entwicklungsumgebung und Versionierung

In der Vorbereitung auf die Programmierungsphase wurde festgestellt, dass es unumgänglich sein wird, dass mehrere Personen zeitgleich an den gleichen Dateien und Projekten arbeiten werden, weshalb die Programmierumgebung für alle Entwickler möglichst gleich sein sollte. Da die Entscheidung für JAVA als Programmiersprache bereits relativ früh fiel, entschied sich die Projektgruppe dafür, als IDE Eclipse zu verwenden, da dieses für eine Vielzahl von Plattformen verfügbar sowie durch Plugins erweiterbar ist. Die IDE und JDK bleiben meistens konstant im Laufe der Entwicklung, wobei Entwickler jederzeit zusätzliche Bibliotheken benötigen können. Aus diesem Grund wurde Maven als ein Build-Management-Tool eingesetzt.
Maven ist ein Build-Management-Tool von der Apache Foundation. Die wichtigste Aufgabe dieses Tools besteht darin, das Leben der Entwickler zu erleichtern, indem viele Schritte z. B. Kompilieren, Testen, Erstellung eines bestimmten Packages automatisiert werden. Um diese Schritte automatisch ausführen zu können, muss eine für Maveninte-gration verantwortliche Person die Projekte, als Maven-Projekte erstellen und einrichten. Um Maven Projekte einzurichten, muss die XML-Datei pom.xml entsprechend der Anforderungen an Bibliotheken, Build und Package angepasst werden. In der pom.xml werden alle Bibliotheken und Quellen für die Bibliotheken, wenn diese gebraucht werden, aufgelistet werden. Außerdem muss der Lebenszyklus des Maven-Builds entsprechend angepasst werden.
Der Maven-Lebenszyklus besteht aus mehreren Phasen:

- validate. Prüfung, ob Projektstruktur gültig und vollständig ist.(Fehlende Bibliotheken werden heruntergeladen)

- compile. Der Quellcode wird kompiliert.

- test. Hier können Tests für den Code hinzugefügt werden.(z.B. jUnit-Tests)

- package. Kompilierter Code und nichtkompilierbare Dateien werden in ein Softwarepaket gepackt.(z.B. Hier kann ein Jar-Paket erstellt werden)
- integration-test. Das Softwarepaket wird auf eine andere Umgebung gepackt und getestet.
- verify. Prüfung, ob das Softwarepaket eine gültige Struktur hat.
- install. Installiert das Softwarepaket in das lokale Maven-Repository, um es in anderen Projekten benutzen zu können.
- deploy. Installiert das Softwarepaket in das entfernte Maven-Repository.(Softwarepaket wird released)

7.2. Bibliotheken

Dieser Abschnitt beschäftigt sich mit den für die Projektgruppe relevanten Programmbibliotheken, die zur Implementierung der SVDI genutzt wurden. Bibliotheken, die lediglich Schnittstellen zu den im vorherigen Kapitel erwähnten Programmen bzw. Komponenten realisieren, werden hier nicht behandelt.

7.2.1. SAGA

Bei SAGA handelt es sich um eine API für verteilte Gridanwendungen. Die SAGA-API bietet dabei Unterstützung für Jobs, Ressourcenüberwachung, sowie Datentransfer und -management. Die SAGA-API verbirgt dabei die Komplexität unterschiedlicher Grid-Middlewares (z.B. Globus Toolkit, gLite, Unicore) vor dem Anwendungsprogrammierer und stellt für diesen eine einheitliche Schnittstelle zur Verfügung. Für die Projektgruppe war hier die Abstraktion des Datentransfers von Interesse und wird zur Abwicklung des Datentransfers über gridftp, gsiftp und FTP genutzt.
SAGA-Implementierungen stehen für eine Vielzahl von Programmiersprachen zur Verfügung, unter anderem als JSAGA in der Version 1.1.1 [1] auch für Java. Bei JSAGA handelt es sich um eine Weiterentwicklung der klassischen SAGA-API, welche von der aktuellen JAVA-Version nicht mehr unterstützt wird. JSAGA gliedert sich in eine Vielzahl von Paketen, von denen die Projektgruppe jedoch nicht alle nutzt, da beispielsweise keine Job-Verarbeitung über die SVDI vorgesehen ist. Bei den genutzten Paketen handelt es sich um das Security-Paket, welches Interfaces für Sessions, Kontextinformationen sowie Zugriffsrechte bereitstellt, das File-Paket, mittels dem Dateien und Ordner manipuliert

[1] http://grid.in2p3.fr/jsaga/saga-apidocs/index.html

werden, das URL-Paket für Zugriffe mittels URLs sowie das Error-Handling-Paket und das Namespace-Paket. [sag] JSAGA realisiert einige wichtige Funktionalitäten der SVDI:

- Upload und Download von Dateien (FTP-Server, Grid) über den SVDI-Client
- Anlegen neuer Ordner auf entfernten Ressourcen (FTP-Server, Grid)
- Umbenennen von Dateien auf entfernten Ressourcen (FTP-Server, Grid)
- Abfragen der System-Metadaten auf entfernten Ressourcen (FTP-Server, Grid)
- Löschen von Dateien auf entfernten Ressourcen (FTP-Server, Grid)

7.2.2. myProxy

Da die Identifikation der Benutzer eines Grids in der Regel mittels Zertifikaten erfolgt, musste auch dieser Authentifikationsmechanismus von der Projektgruppe berücksichtigt werden. Zur Verwaltung der Zertifikate wird von der SVDI die myProxy-Bibliothek benutzt, die hierfür wichtige Funktionalitäten bereitstellt. Im Rahmen der Implementierung haben wir uns für die Nutzung von jglobus-core.jar und myproxy.jar, myproxy-logon.jar entschieden. Diese Bibliotheken sind frei verfügbar zum Download über `http://grid.ncsa.illinois.edu/myproxy/`. Am wichtigsten ist wohl die Erstellung eines aus einem X509-Zertifikats abgeleiteten Proxy-Zertifikats, das über eine beschränkte Lebenszeit verfügt und zur Authentifikation bei Grids genutzt werden kann. Diese Bibliothek wird dabei im SVDI-Client genutzt, um diese Proxy-Zertifikate zu erzeugen, zu speichern und zu löschen, wenn sie nicht mehr benötigt werden. Des Weiteren unterstützt myProxy auch die Nutzung eines Webservices, der Proxy-Zertifikate auf einem Webserver speichert, damit ein Anwender dieser von unterschiedlichen Orten aus nutzen kann. Dieses Feature wird von der SVDI zur Zeit nicht genutzt, würde für die Zukunft jedoch eine interessante Erweiterung darstellen.[myp]

8. Evaluierung

In diesem Kapitel wird beschrieben, was die in der Projektgruppe entwickelte Software leisten kann. Außerdem wird darauf eingegangen, in wie weit das Endergebnis den gestellten Anforderungen und entwickelten Szenarien entspricht.

8.1. Vergleich mit Anforderungen

In der folgenden Tabelle werden Muss- und Kann-Kriterien aufgelistet und mit dem entwickelten Programm verglichen.

Kriterium	kurze Beschreibung	Erfüllt?	Bemerkung
M1	Virtual Storage Layer muss implementiert werden	ja	
M2	Schnittstellendokument	ja	
M3	Datenverwaltung jedes einzelnen Benutzers	ja	
M4	Suchfunktion	ja	Gesucht wird nur nach Metadaten
M5	Einzelne Dateien aus dem Set von Daten auslesen	ja	
M6	Metadaten für die Daten	ja	
M7	Metadatendienst	ja	
M8	Operationen für Metadaten: Anlegen, Löschen und Ändern	ja	
M9	Redundanz der Metadaten	ja	MetadatenDB (MongoDB) kann auf Cluster betrieben werden

M10	Metadatensuche und Einschränkung der Ergebnisliste	teilweise	keine Einschränkungen der Ergebnisliste auf dem Client, weil Suche auf dem Server implementiert ist und Suchergebnisse direkt auf dem Client angezeigt werden.
M11	Datenquellen einsehen, Informationen über Ressource beziehen, Zugriffe kontrollieren, Änderungen durchführen und Daten aus verschiedenen Speicherquellen beziehen	teilweise	Keine Kontrolle über Zugriffsrechte. Für mehrere unterschiedliche Ressourcen ist dies zu aufwendig
M12	Zugriffsrechtekonzept	nein	Keine Dateifreigaben für andere Benutzer (s. M11)
M13	Unterstützung der Basisszenarien	ja	
M14	Benutzerfreundlichkeit	ja	
M15	Unterstützung unterschiedlicher Datenaufbewahrungstechnologien	ja	Unterstützt werden: FTP, SFTP, Grid, Lokal
M16	Dokumentation	ja	
M17	Plattformunabhängigkeit der Software	ja	
M18	Unterstützung der TU Ressourcen	ja	

Tabelle 8.1.: Muss-Kriterien

Kriterium	kurze Beschreibung	Erfüllt?	Bemerkung
K1	Anbindung verschiedener Grids und Clouds	ja	Keine Anbindung für Cloud
K2	Einbindung lokaler Dateisystem	ja	
K3	API für externe Programme	ja	Webserver bietet Webservices an, die von anderen Anwendungen benutzt werden können

K4	Plattformunabhängiger Client	ja	Da der Client in Java implementiert ist, kann dieser unter verschiedenen Betriebssystemen funktionieren (mobile Betriebssysteme sind ausgeschlossen)
K5	Backups	nein	
K6	Einstellungen zentral speichern	ja	Programmtechnisch wurde die Funktionalität implementiert
K7	Anbindung von Amazon S3, Dropbox etc.	nein	
K8	Verschlüsselung bei Datenhaltung und Datentransfer	nein	Datenübertragung verschlüsselt mittels HTTPS
K9	Caching-Methoden für performantes Arbeiten	teilweise	Clientseitiges Caching zum Teil vorhanden. Gecacht wird z.Z. das Benutzerprofil, welches Informationen über hinzugefügte Ressourcen und bereits heruntergeladene Dateien enthält.
K10	Spezielles Verhalten bei Verbindungsabbrüchen	nein	Ist nicht vorgesehen
K11	Fachtermina im Programm	teilweise	
K12	Internationalisierung	nein	Internationalisierung ist nicht implementiert worden, ist aber möglich
K13	Umfangreiches Rechtemanagementkonzept	nein	

Tabelle 8.2.: Kann-Kriterien

8.2. Vergleich mit Szenarien

In diesem Kapitel wird die entwickelte Software mit den Szenarien verglichen. Die Szenarien basieren auf Anforderungen zur Software und spiegeln das Benutzerverhalten wider. Da die Szenarien aufeinander aufbauen, kann die Anzahl der zu testenden Fälle reduziert werden. Aus den Szenarien wurden die einzelnen Anforderungen an das Programm extrahiert. Dabei handelt es sich um die folgenden Punkte:

- Registrierung eines neuen Benutzers

- Anmeldung mit einem Benutzernamen und einem Passwort
- Einbindung einer Ressource(Benutzerdaten, Zertifikate)
- Mehrere Ressourcen einbinden, die schon Dateien enthalten.
- Datentransfer (Hoch- und Herunterladen) von einem lokalen Rechner zu einer entfernten Ressource und zurück.
- Datentransfer zwischen entfernten Ressourcen.
- Metadaten erstellen und bearbeiten.
- Suche nach Dateien anhand von Metadaten.
- Gesuchte Dateien direkt auf eine andere Ressource kopieren.
- Replikation der Daten.
- Einstellung der Zugriffsrechte für Benutzer.

Die meisten dieser Anforderungen werden erfüllt, da das Programm die entsprechende Funktionalität aufweist. Es gab aber Funktionen, die nicht in vollem Umfang implementiert werden konnten oder deren Beschreibung etwas unklar ist. Darauf wird im Folgenden eingegangen:

Einbindung einer Ressource(Benutzerdaten, Zertifikate)

Da der Begriff „Ressource“ nicht eindeutig ist, musste die minimale Anzahl Ressourcen mit denen die Software arbeiten kann, vorab festgelegt werden. Das Programm sollte mit FTP, SFTP, Grid und Cloud arbeiten können. Wegen technischen Umständen konnte die Anbindung von Clouds nicht implementiert werden. Aus diesem Grund wurden FTP-, SFTP- und Grid-Verbindung evaluiert.

Mehrere Ressourcen einbinden, die schon Dateien enthalten.

Das Dateisystem der angebundenen Ressource sollte angezeigt werden, damit der User bequem mit der Ressource arbeiten kann.

Metadaten erstellen und bearbeiten

Metadaten umfassen ein sehr breites Feld. Aus diesem Grund haben wir zwei Arten von Metadaten (System- und Benutzermetadaten) definiert. Deswegen wurde geprüft, dass Metadaten beider Arten erstellt und bearbeitet werden konnten.

Replikation der Daten

Automatische Replikation der Daten wurde aus technischen Gründen nicht implementiert. Wenn Daten repliziert werden müssen, können diese manuell auf eine eingebundene Ressource hochgeladen werden.

Einstellung der Zugriffsrechte für Benutzer

Da jede Ressource ein eigenes Rechtemanagement bezüglich seiner Benutzer und des Dateisystems besitzt, konnte aus Zeitgründen diese Funktionalität nicht implementiert und somit auch nicht evaluiert werden.

8.3. Korrektheit

Ein wichtiger Teil der Entwicklung ist das Testen der bereitgestellten Schnittstellen und Methoden, da durch das rechtzeitige Testen Fehler früh entdeckt und behoben werden können. Während des Testens sollten folgende Fragen beantwortet werden:

- Macht das Programm das, was es machen soll? (Zuverlässigkeit)
- Liefert das Programm das richtige bzw. erwartete Ergebnis? (Korrektheit im Speziellen)
- Wie verhält sich die Software in Fehlerzuständen? (Robustheit und Fehlerverhalten)
- Werden die richtigen Fehlermeldungen zurückgegeben? (Fehlerbehandlung)

Für das Testen wurden zwei Werkzeuge benutzt. Mittels JUnit wurde der Programmcode getestet. Die Schnittstellen bzw. WebServices wurden mit SoapUI getestet.

8.3.1. JUnit

JUnit ist ein Framework zum Testen von Java-Programmen, das für automatisierte Tests einzelner Klassen oder Methoden geeignet ist. Durch Unit-Tests kann festgestellt werden, ob das Programm das erwartete Ergebnis bzw. den erwarteten Fehler zurück gibt. Außerdem wird der Programmierer auf unerwartete Exceptions hingewiesen. Die JUnit-Tests wurden für alle Klassen und Methoden, die Geschäftslogik beinhalten, geschrieben. Zusätzlich wurde JUnit ins Build-Management-Tool Maven integriert. Somit wurde der Programmcode während des Build-Vorgangs bei jeder Aktualisierung der Software getestet. Der Entwickler konnte somit die Fehler direkt feststellen und nachbessern. Dies hat viel

dazu beigetragen, dass Commits fast keine (erwarteten) Fehler beinhalteten oder diese zumindest noch vor Fertigstellung der Software korrigiert werden konnten. Die Software war somit bei Release insofern fehlerfrei, als dass sämtliche Tests keine Fehler entdecken konnten.[jun]

8.3.2. SoapUI

Bei SoapUI handelt es sich um ein Testframework für Webservices. Da die Kommunikation zwischen dem Server und dem Client unserer Anwendung mittels SOAP erfolgte, wurde ein Tool benötigt, das die SOAP-Nachrichten an den Server schicken und eine Antwort empfangen konnte. SoapUI bietet die dafür erforderlichen Funktionen an:

- Inspektion und Aufruf von Webservices
- Funktionaler Test von Webservices
- Lasttest von Webservices
- Erstellung von Webservice-Dummies für den Test.

Viele dieser Funktionen wurden für das Testen des Servers benutzt. Anfangs wird die WSDL-Datei des Webservices in SoapUI eingebunden. Anhand dessen können schon erste Fehler entdeckt werden (z.B. Server offline, falsche Stubs werden generiert, weil WSDL-Datei fehlerhaft ist). Anschließend kann jede Methode des Servers getestet werden, indem SOAP-Nachrichten erstellt und gesendet werden. SoapUI bietet ausreichend Information über die Antwort des Services, indem die Antwort-SOAP-Nachricht jederzeit angezeigt und gespeichert werden kann. Da SoapUI über eine eigene Script-Sprache zur Erstellung der Nachrichten verfügt, die noch erlernt werden musste, hat der Test-Manager ein Java-Tool geschrieben, das die XML-Datei entsprechend der Webservice-Methoden mit zufälligen Werten generierte. Der Lasttest der Webservices erfolgte, indem eine große Zahl SOAP-Nachrichten gleichzeitig an den Webservice gesendet wurde. Leider konnte der Lasttest aus technischen Gründen nicht so umfangreich ausgeführt werden, wie es wünschenswert gewesen wäre. Dies hing damit zusammen, dass nicht genug Clients und Clientbandbreite zu Verfügung stand und eine Vielzahl von Zugriffen auf den Server innerhalb des Universitätsnetzes eventuell auch Dritte in Mitleidenschaft gezogen hätte. Es wurden daher nur ca. 10 Verbindungen gleichzeitig hergestellt und mehrere SOAP-Nachrichten an jede Methode gesendet. Der Server hat auch diesen Test erfolgreich überstanden.[jun]

9. Fazit

Alles in allem kann man die Projektgruppe der Scalable Virtual Data Infrastructure als einen Erfolg für alle Beteiligten bewerten. Sowohl die Teilnehmer, als auch die Betreuer, konnten durch das Projekt neue Erfahrungen sammeln und ihren Wissensumfang um soziale, technische und zwischenmenschliche Kompetenzen erweitern. Die Betreuer erprobten sehr erfolgreich neue Methoden der Kommunikation und Konfliktlösung auf Basis des problembasierten lernens. Diese Techniken ermöglichten dem Team eine deutliche Verbesserung der Kommunikation untereinander und trug bedeutend zum Projektfortschritt bei. Durch die Verwendung, für die meisten der Teilnehmer unbekannter Entwicklungstechniken aus dem Bereich der Java Enterprise Umgebung, konnten neue Erkenntnisse gesammelt und vertieft werden. Auch auf den Bereich der sicheren Softwareentwicklung wurde großer Wert gelegt. Die Verwendung von Opensource Software für die Datenbanken und Bibliotheken deckte nicht erwartete Probleme neben den geschätzten Vorteilen dieser Software Art auf. Unter Anderem basieren viele Bibliotheken auf freier Software, die von freiwilligen Entwicklern in ihrer Freizeit entwickelt wird. Leider fehlten oftmals viele beschriebene Komponenten oder es mangelte an ausführlicher Dokumentation der Bibliotheken. Daher mussten verschiedene Quellen ausprobiert werden, um eine passende Lösung zu finden. Entsprechend musste die von der Projektgruppe entwickelte Software in großem Umfang angepasst werden. Die Betreuer haben mit ihrer sehr guten Betreuung sowohl auf der technischen Ebene als auch im sozialen Umfeld die Teilnehmer zu Höchstleistungen motiviert und die Projektgruppe zu einem erfolgreichen Projektziel geführt. Alle Teilnehmer mussten in ihren jeweiligen Aufgabenbereichen große Verantwortung übernehmen und konnten sich dadurch einen Überblick über die Wichtigkeit einer funktionierenden Zusammenarbeit verschaffen. Trotz einiger Rückschläge, zu beginn des Projektes, kann daher die Projektegruppe als erfolgreich betrachtet werden. Die Endprodukte der Projektgruppe sind eine funktionierende Software, sowie eine Dokumentation und ein abschließender Endbericht. Jeder Teilnehmer kann durch die Teilnahme an dieser Veranstaltung wertvolles neues Wissen und Erfahrungen, sowie eine verbesserte soziale Kompetenz im Umgang mit großen Projekten, mit in seine zukünftige Laufbahn nehmen.

Literaturverzeichnis

[BF06] BARTOL FILIPOVIÃ$\frac{1}{4}$, Tobias S.: Grid Security Infrastructure - ein Überblick / Fraunhofer-Institut fÃ$\frac{1}{4}$r Sichere Informationstechnologie Rheinstr. 75 64295 Darmstadt. 2006. – Forschungsbericht

[Car11] CARLSON, Marc: *Interoperable Cloud Storage with the CDMI Standard.* presentation. `http://www.snia.org/sites/default/education/tutorials/2011/spring/cloud/MarkCarlson_Interoperable_Cloud_CDMI.pdf`. Version: 2011. – Bilder 4.3, 4.4 zu CDMI, zuletzt abgerufen am 20.11.2011

[EJB] Oracle

[Fos02] FOSTER, Ian: *What is the Grid? A Three Point Checklist.* 2002

[GHJV95] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph E. ; VLISSIDES, John: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995

[gri] *Grid Torrent.* Webpage. `http://www.cslab.ece.ntua.gr/cgi-bin/twiki/view/CSLab/GridTorrent`. – zuletzt abgerufen am 30.08.2012

[HCK+08] HUPFELD, Felix ; CORTES, Toni ; KOLBECK, Björn ; STENDER, Jan ; FOCHT, Erich ; HESS, Matthias ; MALO, Jesus ; MARTI, Jonathan ; CESARIO, Eugenio: The XtreemFS architecture - a case for object-based file systems in Grids. In: *Concurr. Comput. : Pract. Exper.* 20 (2008), Dezember, Nr. 17, 2049–2060. `http://dx.doi.org/10.1002/cpe.v20:17`. – DOI 10.1002/cpe.v20:17. – ISSN 1532–0626

[IF01] IAN FOSTER, Steven T. Carl Kesselman K. Carl Kesselman: The Anatomy of the Grid Enabling Scalable Virtual Organizations. In: *International Journal of High Performance Computing Applications* (2001)

[jun] *jUnit Overview.* webpage. `http://junit.sourceforge.net/doc/cookstour/cookstour.htm`. – zuletzt abgerufen am 30.08.2012

[lus] *Lustre: A Future Standard for Parallel File Systems?* presentation. `http:`

//www.studham.com/scott/files/ISC2005_Lustre_Studham.pdf. – zuletzt abgerufen am 30.08.2012

[Man06] *Kapitel* 1.4.11 Grid Computing, 1.4.9 Arten von Grids. In: MANHART, Dr. K.: http://www.tecchannel.de/netzwerk/wan/439222/networked_computing_grundlagen_und_an 2006

[mon] *MongoDB Overview.* Webpage. http://www.mongodb.org/. – zuletzt abgerufen am 30.08.2012

[myp] *MyProxy Overview.* webpage. http://grid.ncsa.illinois.edu/myproxy/. – zuletzt abgerufen am 30.08.2012

[odb] *ODBMS.* Webpage. http://www.odbms.org/. – zuletzt abgerufen am 30.08.2012

[owl] *OWL 2 Web Ontology Language Document Overview.* spezification. http://www.w3.org/TR/2009/REC-owl2-overview-20091027/. – zuletzt abgerufen am 30.08.2012

[p2p] *P2P Overview.* Webpage, Article. http://www.e-teaching.org/technik/vernetzung/architektur/peer-to-peer/. – zuletzt abgerufen am 30.08.2012

[Par98] PARTSCH, Helmut: *Requirements-Engineering systematisch.* Springer-Verlag Berlin Heidelberg, 1998. – 17f S.

[pos] *PostgreSQL Overview.* Webpage. http://www.postgresql.org/about/. – zuletzt abgerufen am 30.08.2012

[rdf] *Resource Description Framework (RDF) Model and Syntax.* spezification. http://www.w3.org/TR/WD-rdf-syntax-971002/. – zuletzt abgerufen am 30.08.2012

[s3F] *Amazon S3 Functionality.* Webpage. http://aws.amazon.com/de/s3/#functionality. – zuletzt abgerufen am 30.08.2012

[sag] *Saga Overview.* webpage. http://grid.in2p3.fr/jsaga/. – zuletzt abgerufen am 30.08.2012

[sema] *Representing Knowledge in the Semantic Web.* presentation. http://www.w3c.it/talks/2005/openCulture/slide1-0.html. – zuletzt abgerufen am 30.08.2012

[semb] *The Semantic Web stack*. presentation. `http://www.w3c.it/talks/2005/openCulture/slide7-0.html`. – zuletzt abgerufen am 30.08.2012

[SGM+08] SCHOLL, T. ; GUFLER, B. ; MÜLLER, J. ; REISER, A. ; KEMPER, A.: P2P-Datenmanagement für e-Science-Grids / Lehrstuhl für Datenbanksysteme, Technische Universität München, Fakultät für Informatik. 2008. – Forschungsbericht

[SNI11] SNIA: *Information Technology - Cloud Data Management Interface (CDMI)*. September 15, 2011. – Bilder 4.1,4.2 zu CDMI

[spa] *SPARQL 1.1 Query Language*. spezification. `http://www.w3.org/TR/sparql11-query/`. – zuletzt abgerufen am 30.08.2012

[web] W3C World Wide Web Consortium

[xml] W3C World Wide Web Consortium

Glossar und Abkürzungsverzeichnis

CDMI Cloud Data Management Interface. 23

CRUD Operationen auf Dateien: Create, Retrieve, Update, Delete. 22

CRUD Storage Networking Industry Association. 23

DaaS Data Storage as a Service. 22

DHT Distributed Hash Table. 32

iSCSI internet Small Computer System Interface, bietet blockbasierten Dateizugriff über das TCP-Protokoll. 22

JSON JavaScript Object Notation. 24

Peer-to-Peer-Connection Direktverbindung zwischen Rechnern. 32

Torrent Peer-to-Peer-Netz. 32

Abbildungsverzeichnis

Tabellenverzeichnis

A. Serverhandbuch

1 Einleitung

Dieses Handbuch soll dazu dienen, den Leser kurz und verständlich zu erläutern, welche Schritte notwendig sind, um den SVDI-Server einzurichten. Dabei wird davon ausgegangen, dass als Betriebsystem Ubuntu in der Version 11.10 eingesetzt wird. Zwar sind sowohl die SVDI-Server-Anwendung, als auch alle von ihr genutzten Dritt-Anwendungen plattformübergreifend auf Linux-, Unix-, und Windowssystemen verfügbar, im Falle eines anderen Betriebssystems werden sich die benötigten Schritte eventuell unterscheiden.

Dieses Handbuch soll weder einen Überblick über die Aufgaben des Servers im Rahmen der SVDI-Architektur, noch über die Implementierung der Schnittstellen geben. Genaueres zur Architektur wird im Endbericht zur Projekgruppe beschrieben, Informationen zu den Schnittstellen finden sich in einem separaten Schnittstellendokument. Auch wird an dieser Stelle nicht auf den laufenden Betrieb und damit eventuell verbundene Administrationstätigkeiten eingegangen, da es sich bei der aktuellen Implementierung der SVDI um einen Prototypen handelt, der noch nicht in der Praxis erprobt wurde.

2 Vorbereitungen

Der SVDI-Server setzt voraus, dass zuvor Instanzen der beiden Datenbanken „MongoDB“ und „PostgreSQL“ sowie des Application-Servers „Glassfish“ installiert und eingerichtet wurden. Außerdem muss als JDK das Ubuntu Paket „Default-jdk“ (`apt-get install default-jdk`)installiert sein. Der SVDI-Server verwendet die Ports 27334, bitte stellen Sie sicher, dass diese freigegeben bzw. nicht geblockt sind. Sollte dies bereits der Fall sein, kann mit Kapitel 3 fortgefahren werden, ansonsten werden die dazu erforderlichen Schritte im Folgenden kurz erläutert.

2.1 MongoDB

Bei Mongo-DB handelt es sich um eine dokumentenorienterte NoSQL-Datenbank, die vom SVDI-Server für die Verwaltung von Metadaten genutzt wird.

- MongoDB von `http://www.mongodb.org/downloads/` herunterladen und entpacken.
- MongoDB startet über `,,bin/mongod‘‘` (optional Datenordner angeben mit `,,--dbpath Ordner‘‘`). Der Datenordner ist der Ordner, in dem sich die von der Datenbank gespeicherten Daten befinden. Sollte kein Datenordner angegeben werden, so wird ein Standardordner angelegt, zur Erstellung dieses Ordners werden Root-Rechte benötigt.

2.2 PostgreSQL

PostgreSQL ist eine objekt-relationale Datenbank, die der SVDI-Server unter anderen für die Benutzer- und Ressourcenverwaltung benötigt.

- PostgreSQL installieren („sudo apt-get install postgresql“).
- Datenbankuser und Datenbank anlegen:
  1. `sudo -u postgres psql` (Hier sollte das Terminal von der Standardansicht zur Postgres-Umgebung wechseln)
  2. `\password postgres`
  3. Beenden mit `\q` (Um zur Standardansicht zurückzukehren)

Nach erfolgreicher Einrichtung sollte PostgreSQL beim Systemstart selbstständig starten.

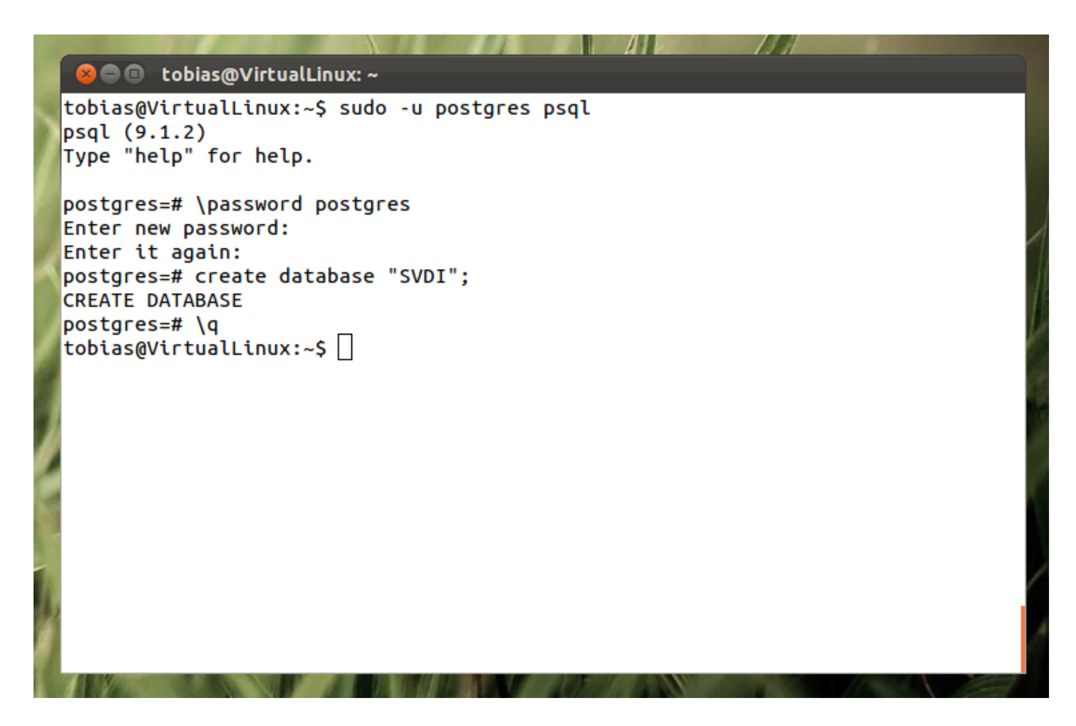


Abbildung 2.1: Einrichten der PostgreSQL-Datenbank

2.3 GlassFish

Bei GlassFish handelt es sich um einen Application-Server, in dessen Umgebung der SVDI-Server läuft und der wichtige Schnittstellendienste für diesen erbringt.

- GlassFish Open Source Edition von `http://glassfish.java.net/public/downloadsindex.html#top` herunterladen und installieren.
- Den JDBC4 Treiber von `http://jdbc.postgresql.org/download.html` herunterladen und in den Glassfish Ordner unter „`glassfish/lib/`“ kopieren.
- Glassfish mittels „`bin/asadmin restart-domain domain1`“ im Glassfish Ordner neu starten. (Allgemein „`bin/asadmin start-domain domain1`“ und „`bin/asadmin stop-domain domain1`“ um die Anwendung zu starten bzw. anzuhalten)

3 Inbetriebnahme des SVDI-Servers

Wenn alle nötigen Vorbereitungen bezüglich der Datenbanken und des Anwendungsservers abgeschlossen sind und diese laufen, kann der SVDI-Server in Betrieb genommen werden. PostgreSQl startet bei Systemstart automatisch, MongoDB und GlassFish müssen selbstständig gestartet warten. Dazu muss zunächst eine Datenbankanbindung geschaffen werden, bevor der eigentliche SVDI-Server auf den Anwendungsserver geladen wird.

3.1 JDBC (Java Database Connectivity)

- mongo-*.jar, morphia-*.jar und den postgresql-jdbc-Treiber in den lib-Ordner im GlassFish-Verzeichnis kopieren (`https://github.com/downloads/mongodb/mongo-java-driver/mongo-2.7.3.jar` und `http://morphia.googlecode.com/files/morphia-0.99.jar`)
- Die Glassfish Administrator-Oberfläche im Browser öffnen (`http://ServerIP:4848/common/index.jsf`).
- Unter „`Resources->JDBC->JDBC Connection Pools`" auf „New" klicken um einen Connection-Pool mit folgenden Daten anzulegen:
  1. Pool Name: Beliebig (z.B. "`SVDIPool`")
  2. Resource Type: „`java.sql.Driver`"
  3. Database Driver Vendor: „`Postgresql`"
- Im nächsten Schritt die folgenden Einstellungen vornehmen:
  1. Pool Settings: Nach Bedarf
  2. Transaction: Die Standardeinstellungen beibehalten
  3. Bei „Additional Properties" müssen die drei vorgeschlagenen Werte ausgewählt und wie folgt belegt werden:
     a) URL: JDBC Connection String in der Form „`jdbc:postgresql://ServerIP/DatenbankName`"
     b) User: Der Datenbenkbenutzer (In diesem Beispiel „`postgres`")
     c) Password: Das Passwort für den gewählten Datenbankbenutzer
- Unter „`Resources->JDBC->JDBC Resources`" auf „New" klicken um eine neue JDBC-Ressource mit den folgenden Daten anzulegen:

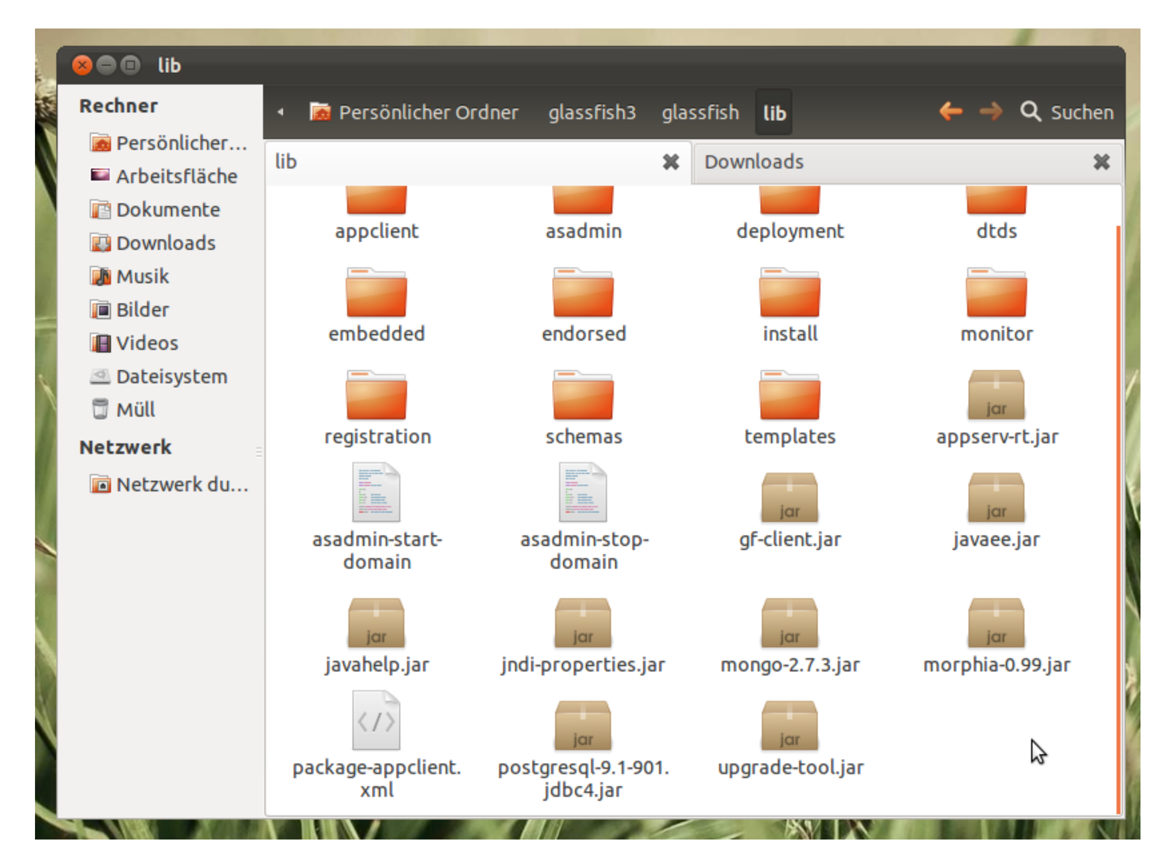


Abbildung 3.1: Die benötigten Dateien im GlassFish-Ordner

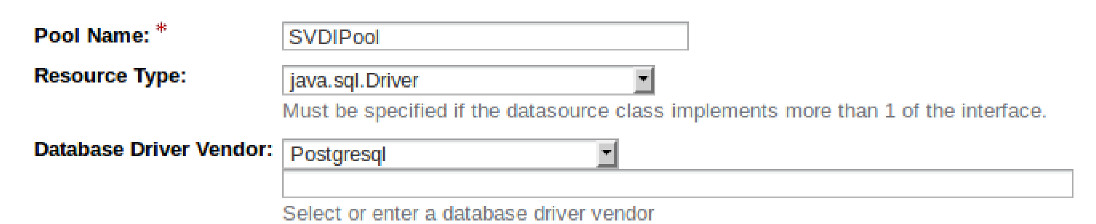


Abbildung 3.2: Einstellungen unter „Connection Pool“

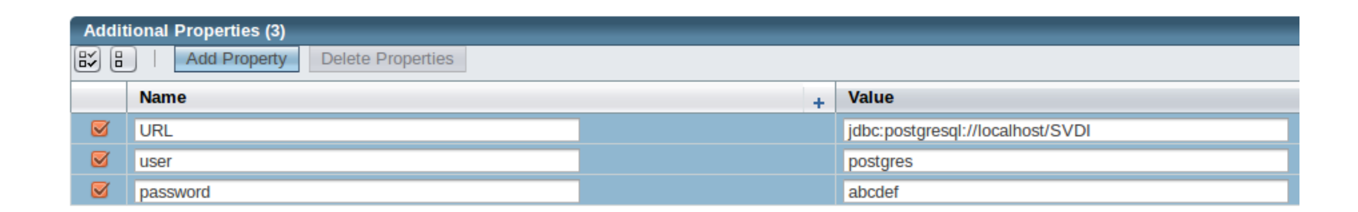


Abbildung 3.3: Weitere Einstellungen

1. JNDI Name: „`jdbc/svdidb`“
2. Pool Name: Den vorher erstellten Pool auswählen (in diesem Beispiel „`SVDIPool`“)

3.2 Deployment

- Die Glassfish Administrator Oberfläche im Browser öffnen (`http://ServerIP:4848/common/index.jsf`).
- Unter „Applications“ auf „Deploy“ klicken und die „`ear-0.1-SNAPSHOT.ear`“ zum Upload auswählen.

B. Schnittstellendokument

1 Einleitung

Dieses Schnittstellendokument dient dazu, dem Leser alle Funktionen, die serverseitig für die Clientanbindung erforderlich sind, an einer Stelle übersichtlich zu präsentieren. So soll gewährleistet sein, dass Dritte in der Lage sind, sowohl den SVDI-Client, als auch den SVDI-Server weiter an ihre Bedürfnisse anzupassen, ohne dabei durch Kompatibilitätsprobleme deren Funktionsfähigkeit zu gefährden.
Das Dokument gliedert sich in vier Teilabschnitte, die thematisch einzelnen Aspekten der SVDI-Architektur (Benutzer-, Ressourcen-, Datei- und Metadatenverwaltung) zugeordnet sind. Diese vier Bereiche wurden als eigenständige Webservices implementiert und sind weitestgehend modular und voneinander unabhängig. Zugleich dienen sie als Anknüpfungspunkt, von dem aus die Geschäftslogik, und Implementierung von Server und Client betrachtet werden können.
Die meisten Webservices verfügen dabei über wenige, jedoch mächtige Schnittstellenmethoden, an die aggregierte Informationen in Form von DTOs (Data Type Objects) übergeben werden. Aufgabe des Clients ist die Ermittlung der benötigten Informationen sowie die Erstellung der DTOs, der Server nimmt diese entgegen und zerlegt sie in kleinere „Pakete“, die anschließend, abhängig von der aufgerufenen Schnittstellenfunktion und den im DTO enthaltenen Daten, an spezialisierte Methoden übergeben werden.
Die Beschreibung der Webservices beginnt mit einem kurzen Überblick über deren Funktionalität, wobei genaueres im Endbericht zur Projektgruppe zu finden ist. Es folgen die dort vorhandenen Schnittstellenfunktionen (Funktionsheader, Aufgabe und Voraussetzungen) sowie eine Beschreibung der benötigten DTOs sowie der geworfenen Exceptions.

2 Benutzerverwaltung

Der UserManagement-Webservice ist für die Benutzerverwaltung der SVDI zuständig. Er verwaltet die Zugangsberechtigungen zur Software und stellt durch weitere Verweise, wie zum Beispiel auf Ressourcen, Verbindungen zu weiteren Informationen über den Benutzer her. Bevor ein Benutzer die SVDI verwenden kann, muss er sich zuerst registrieren und anschließend einloggen. Der Zugriff auf sämtliche Dienste der SVDI ist nur während einer gültigen Sitzung, also in der Zeit zwischen Login und Logout möglich.

```
1 @WebMethod
2 public UserDTO register(
3     @WebParam(name = "user") UserDTO user)
4 throws InvalidInputException {...}
```

Listing 2.1: Einen neuen Benutzer registrieren

Die Funktion register ermöglicht es einen neuen Benutzer in der SVDI zu registrieren. Dazu muss ein Objekt vom Typ UserDTO mit den Benutzerinformationen versehen werden und an die Funktion register(...) übergeben werden. Nach einer erfolgreichen Registrierung, wird ein Objekt vom Typ UserDTO zurückgegeben, das die ID des neu registrierten Benutzers enthält. Nun kann der Benutzer mittels login(..) angemeldet werden.

```
1 @WebMethod
2 public UserDTO login(
3         @WebParam(name = "user") UserDTO user)
4 throws InvalidInputException {...}
```

Listing 2.2: Einloggen eines Benutzers

Die Funktion login erlaubt es sich an der SVDI anzumelden, um eine neue Sitzung zu beginnen und die SVDI danach verwenden zu können. Eine Anmeldung erfordert korrekte Zugangsdaten durch einen erfolgreichen Registrierungsvorgang. Übermittelt werden die Zugangsdaten auch durch ein Objekt vom Typ UserDTO.

```
1 @WebMethod
2 public UserDTO logout(
3         @WebParam(name = "user") UserDTO user)
4 throws InvalidInputException {...}
```

Listing 2.3: Ausloggen eines Benutzers

Mit der Funktion logout kann sich ein Benutzer von der SVDI trennen. Durch ein Objekt vom Typ UserDTO und den passenden Accountdaten wird der Benutzer getrennt und kann sich danach erneut durch login(...) anmelden.

```
@WebMethod
public UserDTO update(
        @WebParam(name = "user") UserDTO user)
throws InvalidInputException {...}
```

Listing 2.4: Ändern der Benutzerinformationen

Mit der Funktion update können die Benutzerdaten überschrieben werden. So können mit dieser Funktion zum Beispiel das Passwort, die E-Mail-Adresse oder weitere Informationen und Einstellungen geändert werden. Die neuen Accountdaten werden in ein Objekt vom Typ UserDTO geschrieben und anschließend vom Server mit den für die einzelnen Daten vorgesehenen Funktionen weiterverarbeitet.

3 Verwaltung entfernter Ressourcen

Der ResourceManagement Webservice übernimmt die Verwaltung der Ressourcen. Mit Ressourcen sind in diesem Falle entfernte Speicherorte wie Grids, FTP-Server usw. gemeint. Die Ressourcen an sich sind von allen Benutzern verwendbar, und beinhalten noch keinerlei Zugangsdaten. Diese müssen für jeden Benutzer separat hinzugefügt werden, was auch über diesen Webservice gemacht wird.

```
1 @WebMethod
2 public List<ResourceTypeDTO> getAllResourceTypes()
3 {...}
```

Listing 3.1: Unterstützte Protokolle

Die Funktion getAllResourceTypes liefert alle vom SVDI-Server unterstützen Ressourcentypen zurück. Der Client kann damit herausfinden, welche Protokolle der Server beherrscht. Dies ist auch hilfreich, wenn man selbst die Unterstüzung für ein neues Protokoll hinzufügen möchte.

```
1 @WebMethod
2 public ResourceDTO createResource(
3         @WebParam(name = "resource") ResourceDTO resource,
4         @WebParam(name = "user") UserDTO user)
5 throws InvalidInputException{...}
```

Listing 3.2: Definieren einer neuen Ressource

Die Funktion createResource legt eine neue Ressource für alle Benutzer an. Diese Ressource besitzt noch keinerlei Zugangsdaten.

```
1 @WebMethod
2 public List<ResourceDTO> getAllResourcesForAUser(
3         @WebParam(name = "user") UserDTO user)
4 throws InvalidInputException{...}
```

Listing 3.3: Welche Ressourcen nutzt ein User?

Die Funktion getAllResourcesForAUser liefert die Ressourcen zurück, für die der übergebene Benutzer Logininformationen besitzt. Dies kann im Client dazu dienen dem Benutzer eine Liste von Ressourcen zu zeigen, die er aktiv nutzt.

```
1 @WebMethod
2 public boolean deleteResource(
3         @WebParam(name = "user") UserDTO user,
```

```
4   @WebParam(name = "resource") ResourceDTO resource)
5 {...}
```

Listing 3.4: Zugriffsinformationen eines Users löschen

Die Funktion deleteResource löscht alle Logininformationen, die ein Benutzer für diese Ressource besitzt.

```
1 @WebMethod
2 public ResourceDTO getResourceByIdAndUser(
3         @WebParam(name = "user") UserDTO user,
4         @WebParam(name = "resource") ResourceDTO resource)
5 {...}
```

Listing 3.5: Liefert ein Ressourcen-DTO zu einer gegebenen ID und eines Users

Die Funktion getResourceByIdAndUser liefert zu einem gültigen UserDTO und einer gültigen ID einer Ressource ein vollständig ausgefülltes Ressource-DTO an den Client.

```
1 @WebMethod
2 public boolean updateResource(
3         @WebParam(name = "resource") ResourceDTO resource)
4         {...}
```

Listing 3.6: Ändern von Informationen zu einer Ressource

Die Funktion updateResource ändert die Informationen einer Ressource, wie z.B. URL oder Typ.

```
1 @WebMethod
2 public boolean createLoginForUser(
3         @WebParam(name = "login") LoginDTO login)
4 throws InvalidInputException {...}
```

Listing 3.7: Logininformationen erzeugen

Die Funktion createLoginForUser erzeugt die Logininformationen für einen Benutzer zu einer Ressource.

```
1 @WebMethod
2 public boolean updateLogin(
3         @WebParam(name = "login") LoginDTO login,
4         @WebParam(name = "user") UserDTO user)
5 throws InvalidInputException, NotSufficientRightsException
    {...}
```

Listing 3.8: Logininformationen ändern

Die Funktion updateLogin ändert die Logininformationen zu einem User.

4 Dateiverwaltung

Der FileManagement-Webservice stellt alle Funktionen bereit, die für die Verwaltung von Dateien nötig sind. Das betrifft das Erstellen, Löschen, Ändern, Umbenennen und Verschieben von Dateien. Er beschäftigt sich jedoch nicht mit den mit einer Datei verbundenen Metadaten, hierfür ist der Metadaten-Webservice zuständig.

```
1 @WebMethod
2 public boolean createFile(
3         @WebParam(name = "user") UserDTO user,
4         @WebParam(name = "data") @XmlMimeType("application/
             octet-stream") DataHandler data,
5         @WebParam(name = "metadata") FileMetaDataDTO metadata
             )
6 throws NoConnectionToResourceException, NotExpectedException
    {...}
```

Listing 4.1: Erzeugen einer Datei

Mit der createFileFFunktion ist es möglich eine Datei auf einer Ressource zu erzeugen. Dafür nötig sind ein Metadatum zu dieser Datei, das die benötigten Grundinformationen beinhaltet, und der initiale Inhalt der Datei.

```
1 @WebMethod
2 public boolean deleteFile(
3         @WebParam(name = "user") UserDTO user,
4         @WebParam(name = "metadata") FileMetaDataDTO metadata
             )
5 throws NotSufficientRightsException, MetadataTooOldException,
     NoConnectionToResourceException, NotExpectedException,
    InvalidInputException {...}
```

Listing 4.2: Löschen einer Datei

Die Funktion deleteFile löscht eine Datei, die auf einer Ressource liegt.

```
1 @WebMethod
2 public boolean renameFile(
3         @WebParam(name = "user") UserDTO user,
4         @WebParam(name = "metadata-old") FileMetaDataDTO
             metadata_old,
5         @WebParam(name = "data") @XmlMimeType("application/
             octet-stream") DataHandler data,
```

```
6           @WebParam(name = "metadata-new") FileMetaDataDTO
               metadata_new)
7 throws NotSufficientRightsException, MetadataTooOldException,
     NoConnectionToResourceException, InvalidInputException,
    NotExpectedException, IOException {...}
```

Listing 4.3: Umbenennen einer Datei

Die Funktion renameFile benennt eine Datei um. Das Verschieben einer Datei ist mit dieser Funktion nicht möglich. Dafür gibt es die separate Funktion MoveFile. Der alte und neue Name der Datei wird dabei den Parametern metadata_old und metadata_new entnommen.

```
1 @WebMethod
2 public boolean updateFile(
3           @WebParam(name = "user") UserDTO usr,
4           @WebParam(name = "data") @XmlMimeType("application/
               octet-stream") DataHandler data,
5           @WebParam(name = "metadata") FileMetaDataDTO metadata
               )
6 throws MetadataTooOldException, NotSufficientRightsException,
     NoConnectionToResourceException, InvalidInputException,
    NotExpectedException {...}
```

Listing 4.4: Ändern einer Datei

Diese Funktion updateFile ermöglicht es den Inhalt einer Datei zu verändern. Die Datei wird dabei komplett überschrieben.

```
1 @WebMethod
2 public boolean copyFile(
3           @WebParam(name = "metadata-old") FileMetaDataDTO
               origMeta,
4           @WebParam(name = "metadata-new") FileMetaDataDTO
               newMeta,
5           @WebParam(name = "user") UserDTO user)
6 throws InvalidInputException, NotSufficientRightsException,
    MetadataTooOldException, NotExpectedException,
    NoConnectionToResourceException {...}
```

Listing 4.5: Kopieren einer Datei

Die Funktion copyFile erzeugt eine Kopie einer Datei an einem gewünschten Ort. Dabei ist es egal, ob die Datei auf derselben Ressource liegt, oder einer anderen.

```
1 @WebMethod
2 public boolean moveFile(
```

```
3           @WebParam(name = "metadata-old") FileMetaDataDTO
               origMeta,
4           @WebParam(name = "metadata-new") FileMetaDataDTO
               newMeta,
5           @WebParam(name = "user") UserDTO user)
6 throws InvalidInputException, NotSufficientRightsException,
     MetadataTooOldException, NotExpectedException,
     NoConnectionToResourceException {...}
```

Listing 4.6: Verschieben einer Datei

Die Funktion moveFile verschiebt eine Datei von einem Ort an einen anderen. Dies ist unabhängig davon ob die Datei über Ressourcengrenzen verschoben wird, oder auf derselben Ressource bleibt.

5 Metadatenverwaltung

Der Metadaten-Webservice bietet sämtliche Funktionen zur Verwaltung der Metadaten an. Dies umfasst das Erzeugen, Abfragen, Ändern und Löschen von Metadaten. Metadaten gibt es in zwei Formen. Das eine sind Metadaten, die sich auf Dateien beziehen und über die Klasse „FileMetaData“ definiert sind. Das andere sind Metadaten, die sich auf Kategorien beziehen und über die Klasse „CategoryMetaData“ definiert sind. Entsprechend existieren separate Funktionen für diese beiden Formen in der Art „getFileMetaData()“ bzw. „getCategoryMetaData()“.

```
1  @WebMethod
2  public List<FileMetaDataDTO> getFileMetaData(
3          @WebParam(name = "filemeta") FileMetaDataDTO dto,
4          @WebParam(name = "filter") List<MetaDataFilter>
              rangeFilter)
5  throws InvalidInputException {...}
6
7  @WebMethod
8  public List<CategoryMetaDataDTO> getCategoryMetaData(
9          @WebParam(name = "catmeta") CategoryMetaDataDTO dto,
10         @WebParam(name = "filter") List<MetaDataFilter>
              rangeFilter)
11 throws InvalidInputException {...}
```

Listing 5.1: Getter des Metadaten-Webservices

Die Funktionen getFileMetaData und getCategoryMetaData dienen zum Abfragen von Metadaten. Sie geben eine Liste von Metadaten-Objekten zurück, die den über die Parameter übergebenen Bedingungen entsprechen. Der erste Parameter ist ein FileMetaDataDTO- bzw. CategoryMetaDataDTO-Objekt. In diesem Objekt können die Felder gesetzt werden, in denen die zurückgegebenen Metadaten übereinstimmen müssen. Ist z.B. der Owner gesetzt, so werden nur Metadaten zurückgegeben, die ebenfalls diesen Owner eingetragen haben. Bei mehreren gesetzten Feldern, müssen alle Felder übereinstimmen. Sind z.B. Owner und Category gesetzt, so werden alle Metadaten zurückgegeben, die ebenfalls diesen Owner haben und in dieser Category sind. Der zweite Parameter ist eine Liste zusätzlicher Filter. Die Filter werden angewendet um Felder, die einen Zahlenwert beinhalten (z.B. Dateigrößen, Datum), neben der Gleichheit auch auf Größer-als und Kleiner-als zu prüfen. Ein Filter ist ein MetaDataFilter-Objekt, und besteht aus einer Zeichenkette und einer Zahl. Die Zeichenkette muss in der Form „Feldname Operator“ sein. Für Feldname wird der Name des Feldes der Metadaten verwendet, als Operator

stehen „<“, „<=“, „>“, „>=“ zu Verfügung. Ein komplettes Objekt der Form [„size >=“, 100] würde z.B. alle Metadaten liefern, die eine Dateigröße von 100 oder größer haben. Durch die Verwendung mehrerer solcher Filter lassen sich natürlich auch Zahlenbereiche eingrenzen. Ein Filter der Form [„size >=“, 100] und einer der Form [„size <=“, 200] würde nur noch alle Metadaten liefern, die eine Dateigröße zwischen 100 und 200 haben.

```java
@WebMethod
public void setFileMetaData(
        @WebParam(name = "filemeta") FileMetaDataDTO dto)
{...}

@WebMethod
public void setCategoryMetaData(
        @WebParam(name = "catmeta") CategoryMetaDataDTO dto)
{...}
```

Listing 5.2: Setter des Metadaten-Webservices

Die Funktionen setFileMetaData und setCategoryMetaData erlauben das Erzeugen und Ändern von Metadaten. Als Parameter wird das komplette neue Metadata-Objekt übergeben. Die in diesem Objekt gesetzte ID entscheidet darüber, ob das Metadatum neu angelegt, oder ein vorhandenes überschrieben wird. Ist die ID gar nicht gesetzt, oder gibt es noch kein Metadatum mit dieser ID, so wird es angelegt. Existiert ein Metadatum mit der gesetzten ID, so wird dieses durch das übergebene Objekt ersetzt.

```java
@WebMethod
public void deleteMetaData(
        @WebParam(name = "id") String id)
{...}
```

Listing 5.3: Löschen eines Metadatums

Die Funktion deleteMetaData dient dem Löschen eines Metadatums. Als Parameter wird die ID des zu löschenden Metadatums verwendet.

C. Benutzerhandbuch

1 Einleitung

Der SVDI-Client stellt eine Schnittstelle zwischen dem Benutzer und dem ganzen SVDI-System dar. Die volle Funktionalität des SVDI-Systems kann vom Benutzer mittels SVDI-Clients genutzt werden. Es stehen folgende Funktionen zur Verfügung: „Datentransfer“, „Metadatenverwaltung“, „Suche“. Einige dieser Funktionen sind komplex, das heißt, dass sie aus mehreren kleineren Funktionen bestehen, wie zum Beispiel „Metadatenverwaltung“, die aus „Metadaten anlegen“, „Metadaten ändern“und „Metadaten löschen“besteht.

Das fast beliebige Kombinieren der oben genannten Funktionen führt dazu, dass der Benutzer mit dem SVDI-Client eine einheitliche Schnittstelle bezüglich seiner verstreuten Daten bekommt. Diese Schnittstelle hat die die Eigenschaft, dass sie nicht nur die verschiedenen Speicherorte des Benutzers zentral verwalten lässt, sonder auch solcher Funktionen, wie Datentransfer zwischen verschiedenen Typen der Speicherorte der Benutzerdaten ermöglicht. Als ein weiteres nützliches Feature kann die Metadatenverwaltung genannt werden. Sie ermöglicht den Benutzern einerseits die Systemdaten zu entnehmen, andererseits kann der Benutzer seine Daten mit eigenen Metadaten bereichern, was wiederrum zu einer effizienten Suche führt.

Im Kapitel 2 wird das Starten der SVDI-Client erklärt. Kapitel 3 beschäftigt sich mit der Frage, wie sich eine Verbindung einrichten lässt. Außerdem wird hier die zentrale Einheit der SVDI-Client-(Benutzer)-Projekt eingeführt. Im Kapitel 4 wird auf den Datentransfer eingegangen. Anschließend werden in Kapitel 5 Metadaten und Suche angeschaut.

2 Programmstart

Das Client-Programm startet nach dem Ausführen der svdi.jar und zeigt zunächst das Login-Fenster an (siehe 2.1). Sofern Sie bereits ein Benutzerkonto registriert haben, können sie sich mit der von Ihnen angegebenen E-Mail-Adresse sowie dem zugehörigen Passwort beim SVDI-Server anmelden. Sollten Sie noch kein Benutzerkonto besitzen, so klicken Sie bitte auf „Registrieren“.

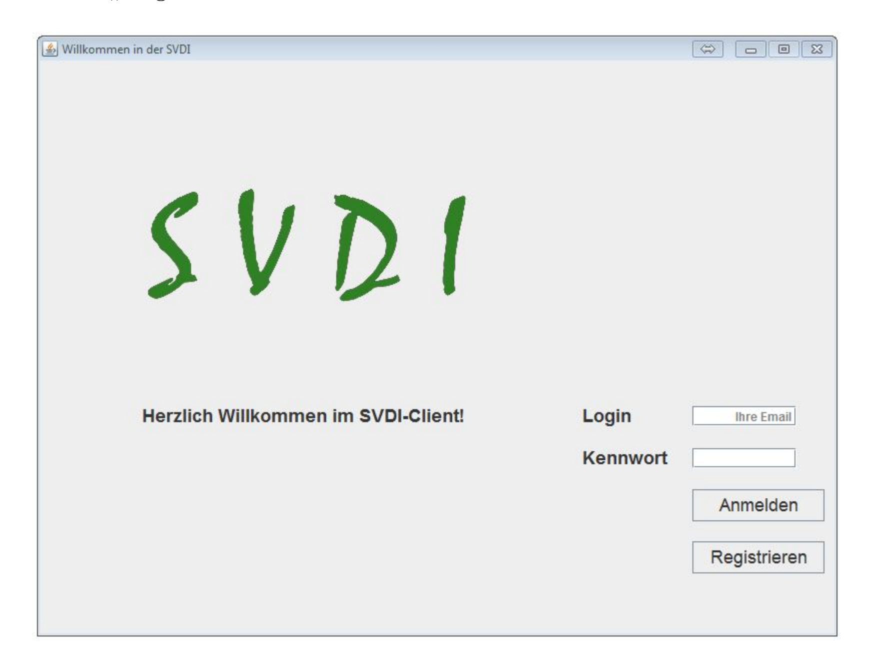


Abbildung 2.1: Der SVDI-Client nach dem Start des Programms

Anrede	Hier wählen Sie bitte zwischen „Herr“ und „Frau“
Titel	Sofern sie einen Titel bzw. akademischen Grad besitzen, können sie diesen hier eintragen
Vorname (erforderlich)	Ihr Vorname
Nachname (erforderlich)	Ihr Nachname
Email (erforderlich)	Ihre Email-Adresse, diese dient gleichzeitig als Name ihres Benutzerkontos
Passwort (erforderlich)	Das von ihnen gewünschte Passwort
Passwort wiederholen (erforderlich)	Das von ihnen gewünschte Passwort

Tabelle 2.1: Erklärung der für die Anmeldung benötigte Informationen

Um ein neues Benutzerkonto einzurichten, werden eine Reihe Informationen von Ihnen benötigt (s. Abbildung 2.2 und Tabelle 2.1):

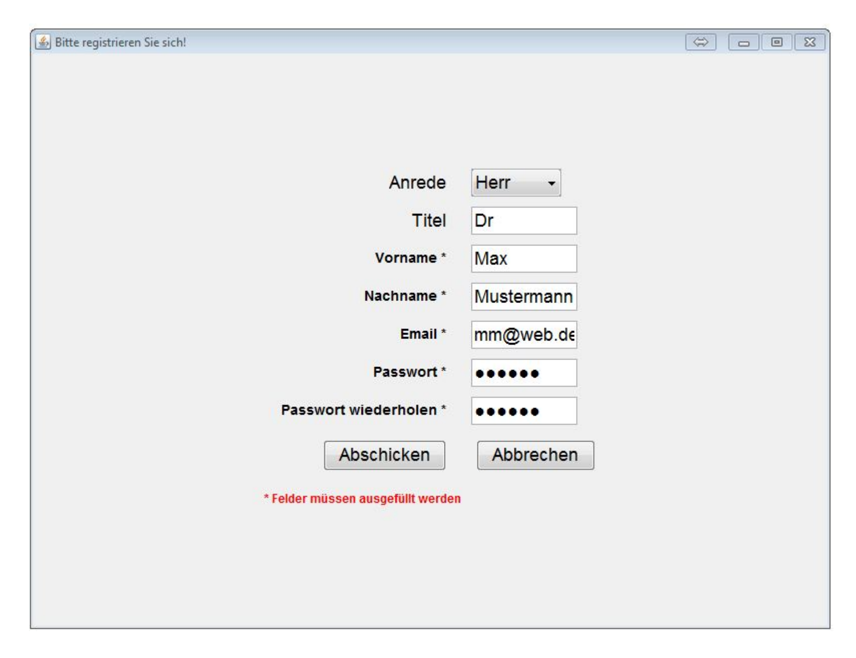


Abbildung 2.2: Registrierung eines neuen Benutzerkontos

Nach erfolgreicher Registrierung können Sie sich mit Ihrer E-Mail-Adresse und dem gewählten Passwort bei der SVDI einloggen.

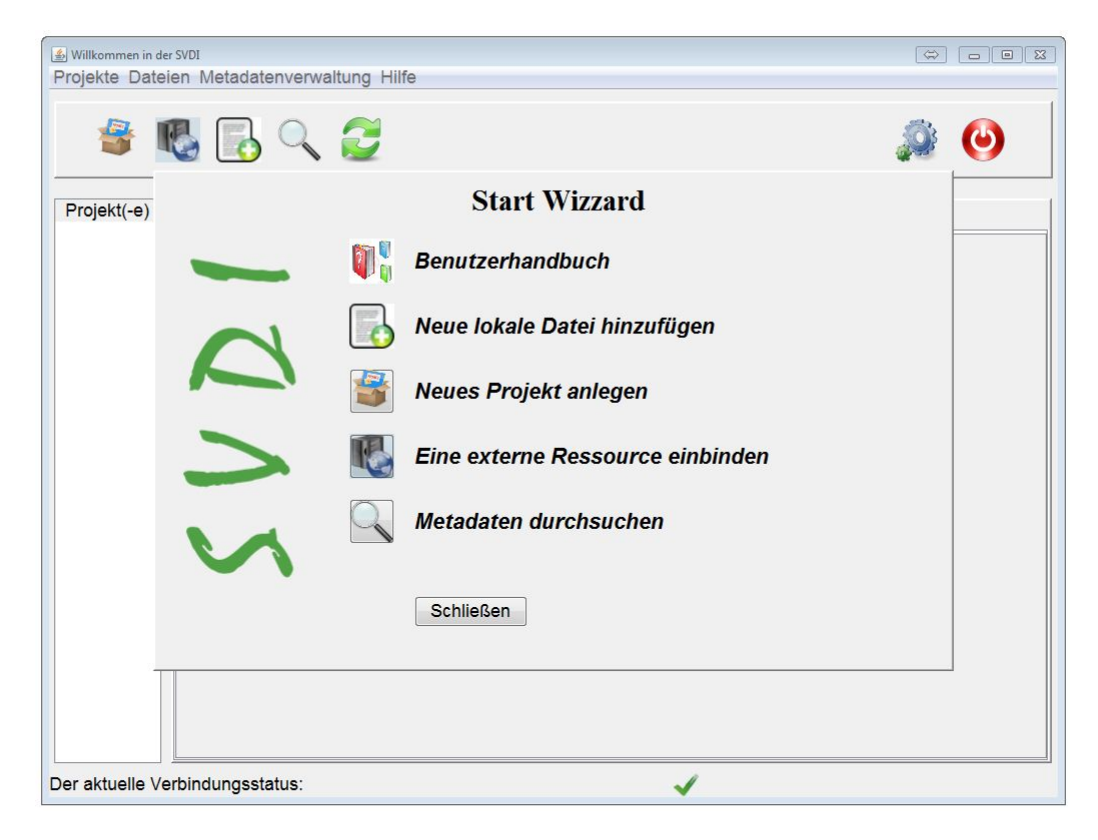


Abbildung 2.3: Das Hauptfenster nach dem Programmstart

Nach erfolgter Anmeldung am SVDI-Server öffnet sich das Hauptfenster 2.3 sowie der Start-Wizard, über den einige oft genutzte Aktionen direkt ausgeführt werden können.

3 Einrichten einer Verbindung

Bevor Sie mit der SVDI arbeiten können, müssen Sie mindestens ein Projekt anlegen, in dem die von ihnen verwendeten Dateien verwaltet werden. Dazu klicken Sie bitte entweder im Start-Wizard oder in der Menüleiste auf den Button „Neues Projekt anlegen“. Daraufhin öffnet sich ein neues Fenster, in dem Sie einen eindeutigen Namen für das Projekt festlegen müssen.

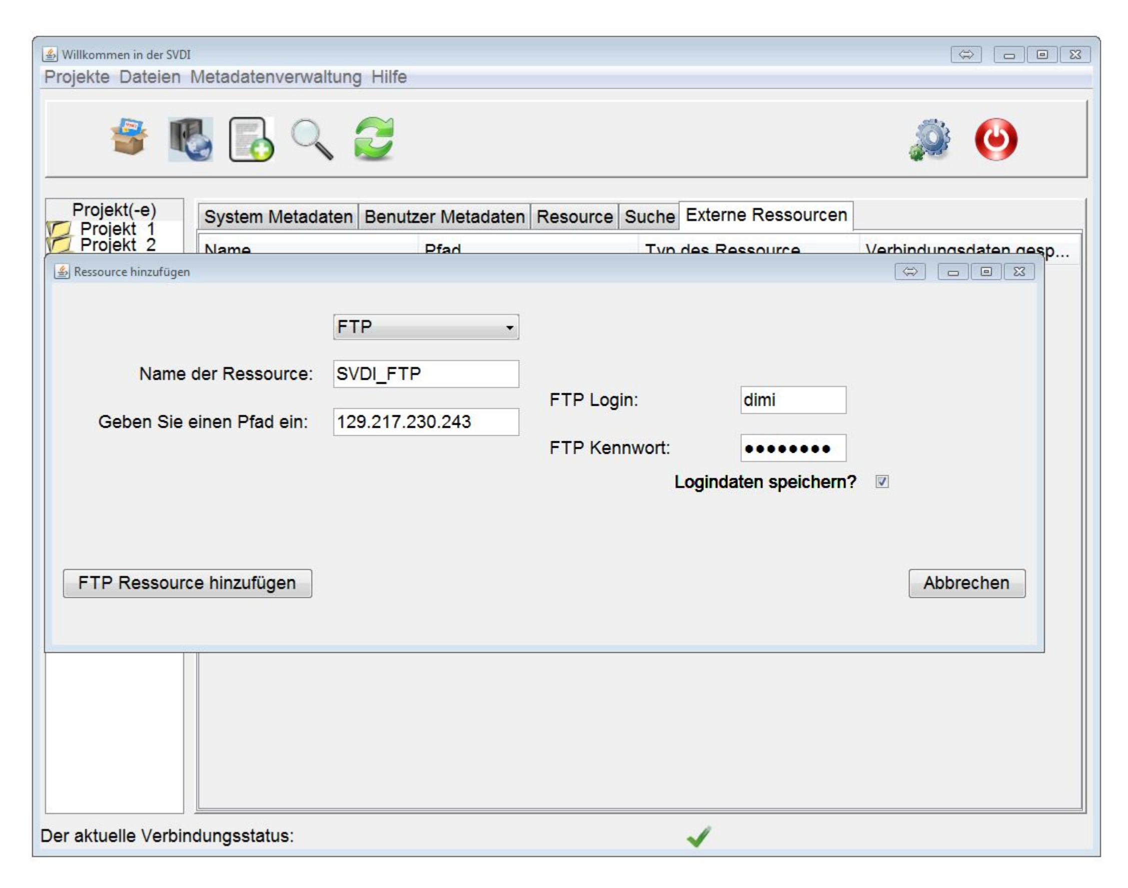


Abbildung 3.1: Hinzufügen einer FTP-Ressource

Des Weiteren muss die SVDI die von ihnen verwendeten externen Ressourcen kennen, um die dortigen Dateien verwalten zu können. Dazu klicken Sie bitte auf den Button „Neue Ressource hinzufügen“ und wählen im Dropdown-Menü den entsprechenden Ressourcentyp aus (zur Zeit werden FTP-,SFTP- und Grid-Ressourcen unterstützt). Ab-

hängig von der gewählten Ressource ändert sich nun die Ansicht. Im Falle einer FTP-Ressource werden die Informationen aus Tabelle ?? benötigt.

Name der Ressource	Der Name, unter dem Sie die Ressource speichern möchten
Pfad	Der Pfad bzw. die IP-Adresse der Ressource
FTP-Login	Ihr Benutzername auf der Ressource
FTP-Passwort	Ihr Passwort auf der Ressource

Der Name der Ressource dient lediglich als Information für Sie und kann beliebig gewählt werden. Sofern sie bei „Logindaten speichern“ einen Haken setzen, merkt sich der SVDI-Client die von ihnen eingegebenen Logininformationen, so dass Sie sie nicht jedesmal neu eingeben müssen, wenn Sie sich mit der Ressource verbinden.

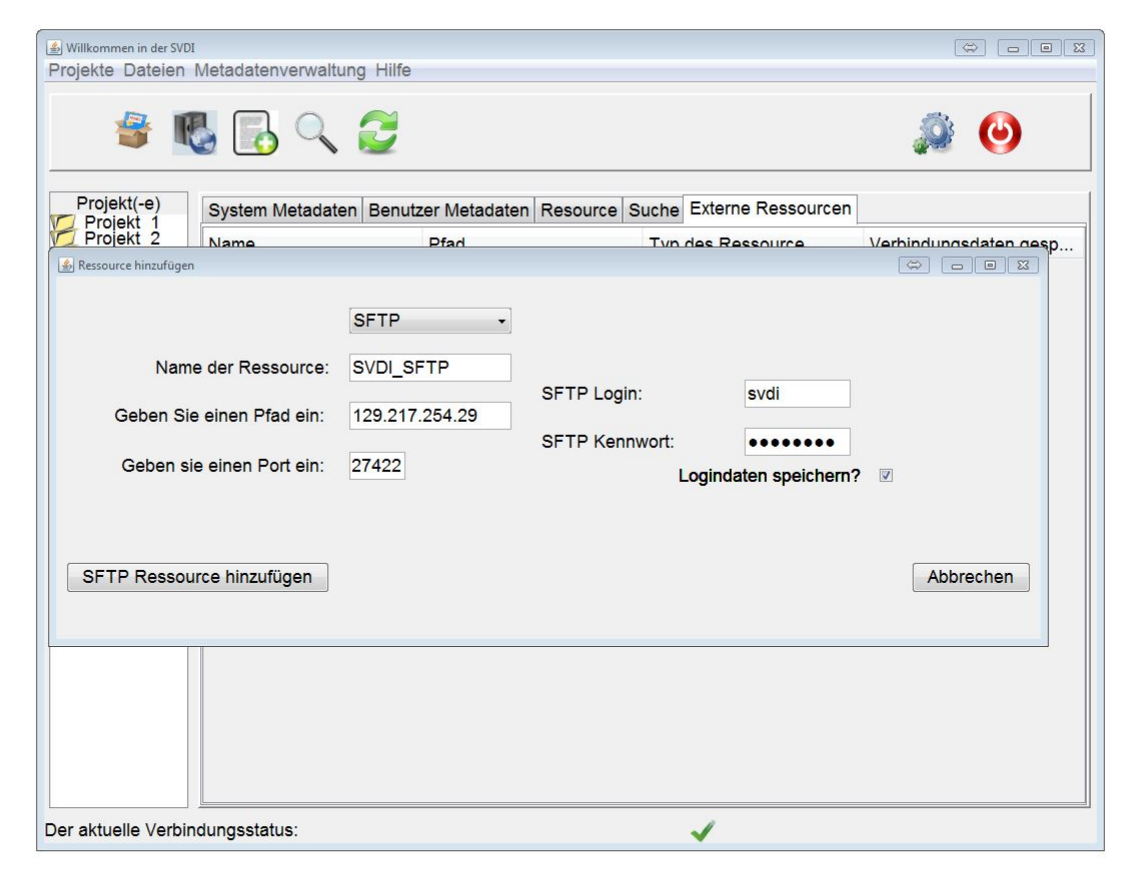


Abbildung 3.2: Hinzufügen einer SFTP-Ressource

Im Falle einer SFTP-Ressource sind die benötigten Informationen nahezu identisch zu

denen einer normalen FTP-Ressource, jedoch muss zusätzlich noch der für die sichere Kommunikation genutzte Port angegeben werden.

Name der Ressource	Der Name, unter dem Sie die Ressource speichern möchten
Pfad	Der Pfad bzw. die IP-Adresse der Ressource
FTP-Login	Ihr Benutzername auf der Ressource
FTP-Passwort	Ihr Passwort auf der Ressource
Port	Der zur Kommunikation genutze Port

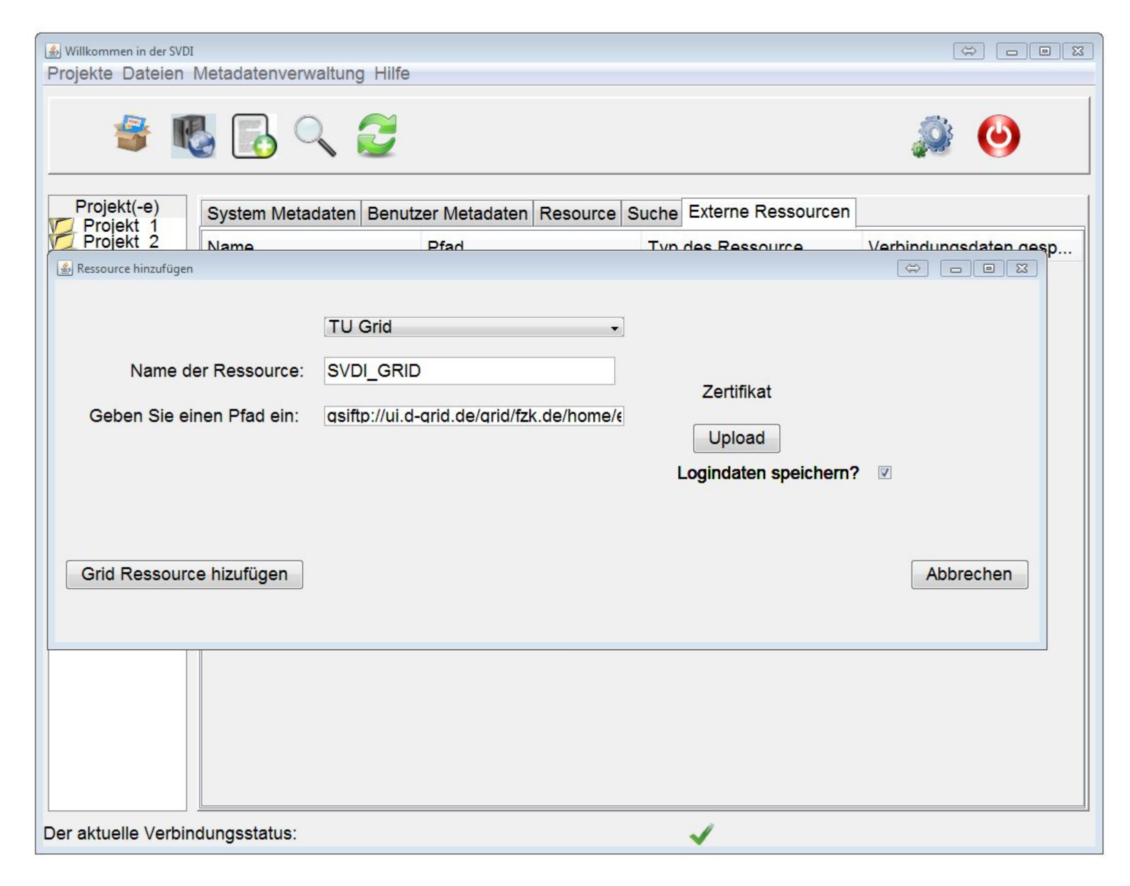


Abbildung 3.3: Hinzufügen einer Grid-Ressource

Im Falle einer Grid-Ressource werden lediglich zwei Informationen benötigt: Der Name sowie der Pfad zur Ressource, wobei Sie hier bitte auch das genutzte Protokoll voranstellen, da sonst keine Verbindung hergestellt werden kann. Das Zertifikat wird direkt aus

dem .globus-Ordner ausgelesen, so dass Sie dieses nicht manuell auswählen müssen.

Name der Ressource	Der Name, unter dem Sie die Ressource speichern möchten
Pfad	Der Pfad der Ressource mit Protokoll

4 Datentransfer

Wie man aus obereb Kapitel entnehmen kann, unterstütz der SVDI-Client 3 Typ der Verbindungen: FTP, SFTP und Grid. Nachdem der Benutzer die Verbindung zur ein oder mehrere Ressource eingerichtet hat, erscheinen sie in einer Tabelle, die in Tab „Externe Ressource“ steht 4.1. Hier lassen sich kurzgefasst einige wichtige Informationen zur gespeicherten Ressource, wie die benutzerdefiniertes Name der Ressource, der Pfad zur Ressource, der Typ der Ressource, sowie der Flag, ob die Verbindungsdaten gespeicher wurden oder nicht, entnehmen.

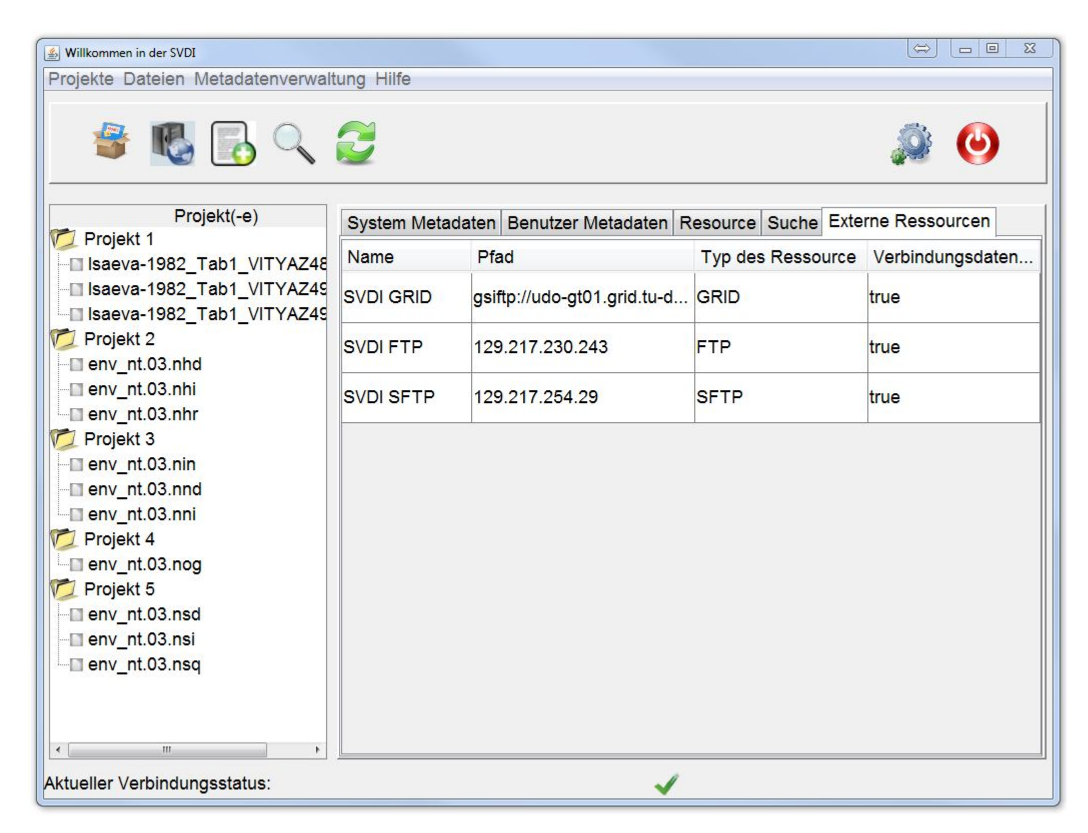


Abbildung 4.1: Ansicht der verfügbaren Ressourcen

Wurden die Verbindungsdaten einer Ressource bei der Einrichtung der Verbindung zur dieser Ressource nicht gespeichert, so werden sie beim Versuch dieser Ressouce aktiv zu schalten gefordert4.2. Ist eingegeben Daten korrekt und ist die Ressource erreichbar, wird der Client automatisch in Tab „Ressource“ wechseln und zur Ressource gehörige Datenbaum anzeigen. Es besteht die Möglichkeit in dieser Datenbaum zu navigieren.

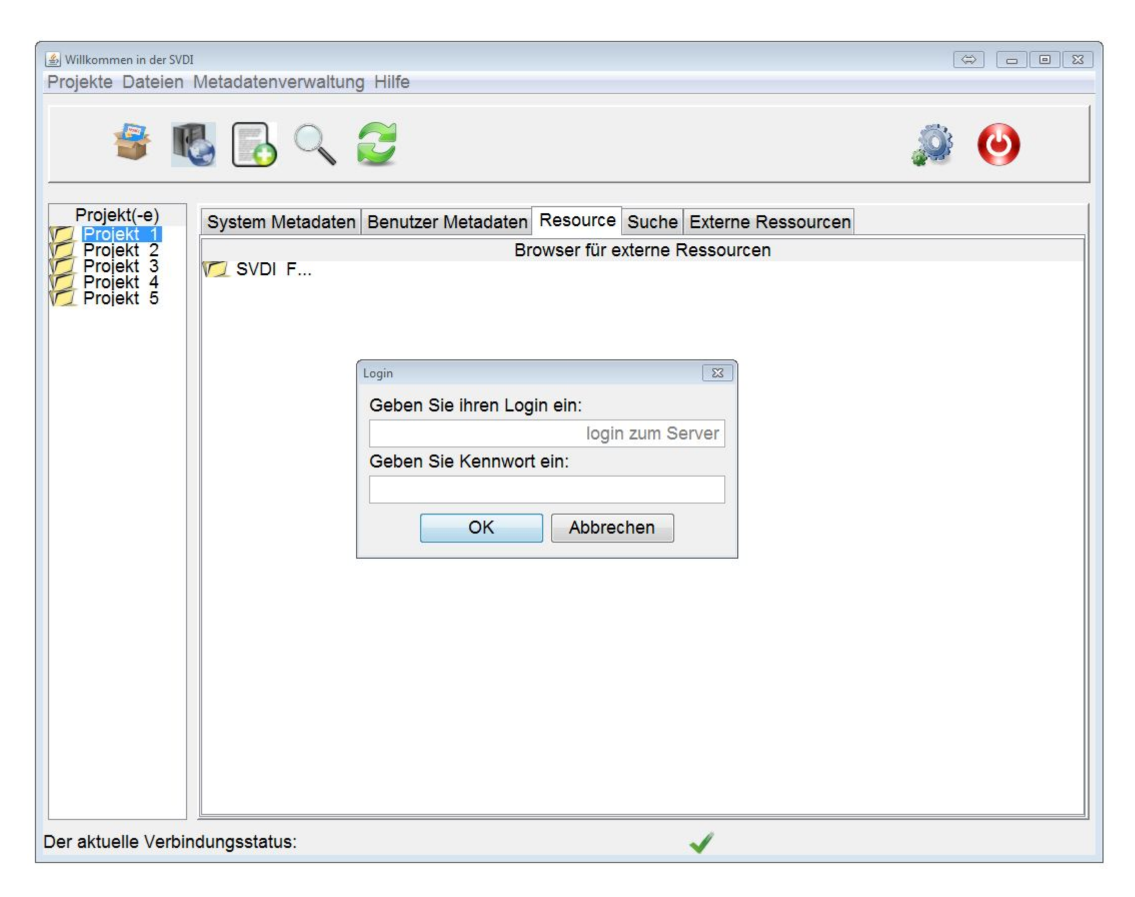


Abbildung 4.2: Benutzerauthentifizierung durch die Ressource

Doppleklick auf Ordner führt dazu, das man in Ordner rein geht. Doppleclick auf Dopplepunkt führt auf eine Ordnerebene nach oben. Doppelclick auf Punkt führt zum Root Ordner der Ressource.

Außer Navigation gibt es die Möglichkeit, die auf dem Ressource ausgewählte Datei zur löschen oder in einer Projekt hinzufügen. Es ist zu beachten, dass eine Datei, die in einer Projekt hinzugefügt wurde, liegt physikalisch immer noch auf dem Ressource. Will man eine echte Kopie der Datei haben, so wählt man die Name der hinzuigefügter Datei in Projekt aus und wählt in Pop-Up Menü den Punkt „Datei herunterladen“.

Als ein Beispiel kann folgendes Szenario angeschaut werden: Man will eine Datei zwischen zwei verschiedener, schon verbundenen, Ressourcen transferieren. Es muss so vorgegangen werden: Erstens aktiviert man die Quellressource, indem man sie in Tabelle aus Tab „Externe Ressource“ auswählt und die Eintrag „Wählen Sie die Ressource aus“ der Pop-Up Menü drückt. Die Datenbaum der Ressource erscheint in „Ressource“-Tab. Jetzt wählt man die zu transferierende Datei aus und fügt sie mit „Die Datei dem Projekt hinzufügen“ in Projekt hinzu. Als nächstes geht man in Projekt, wohin die Datei hinzugefügt wurde und wählt deren Name aus. Mittels Pop-Up Menü Eintrag „Ausgewahlte Datei herunterladen“ wird eine lokale Kopie der Datei in Client hingelegt. Jetzt

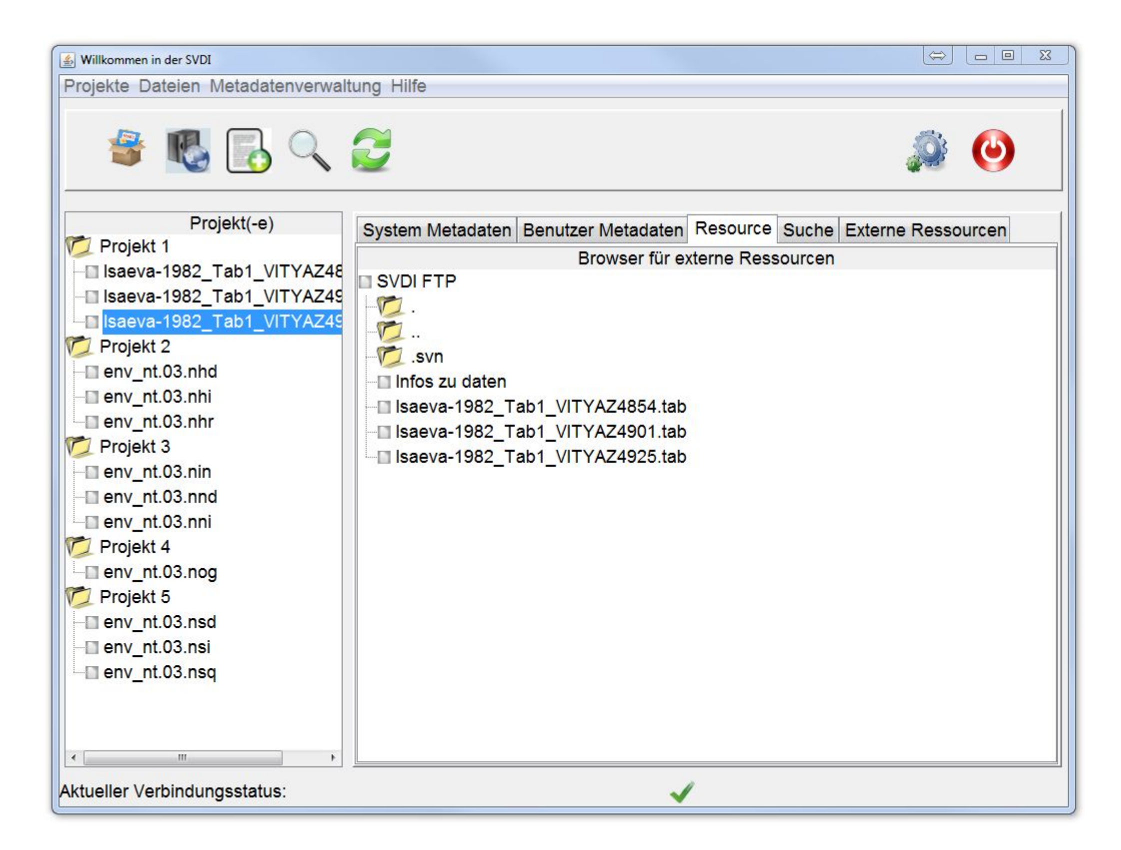


Abbildung 4.3: Dateitransfer auf eine Ressorce

geht man wieder in Tab „Externe Ressource“, wählt die Zielressource aus und bindet sie ein. Danach geht man wieder zur Projekt, wählt die Datei aus und mittels Pop-Up Menü Eintrag „Ausgewahlte Datei hochladen“ ladet man es hoch.

Wie man sieht, es kann auch nur ein Teil der Szenario ausgeführt werden. Also, es ist möglich die Daten aus einer Ressource in ein Projekt herunterladen. Oder andersrum eine lokale Datei aus einer Proejkt auf eine Ressource hochladen. Die Abbildung 4.3 zeigt der Zustand nach dem Datentransfer von drei Dateien aus der Projekt zur einier Ressource.

5 Metadaten

In dieser Kapitel schauen wir die letzte Features, die SVDI-Client hat, nämlich die Metadatenverwaltung und die Suche. Die Metadatenverwaltung bestht aus mehreren Unterfunktionen. Erstens, es können die Systemmetadaten, der in SVDI-Client enthaltene Dateien, entnohmen werden. Man wählt die nötige Datei aus, mit Rechtsklick wird eine pop-Up Menü aufgerufen, dort wählt man „Systemmetadaten anzeigen“ und diese Metadaten werden in Tab „System Metadaten“ sofort angezeigt. Hier steht solcher Information, wie Dateiname, Dateigröße, Name der Ressource, die dieser Datei enthält und die Erstellungsdatum der Datei 5.1.

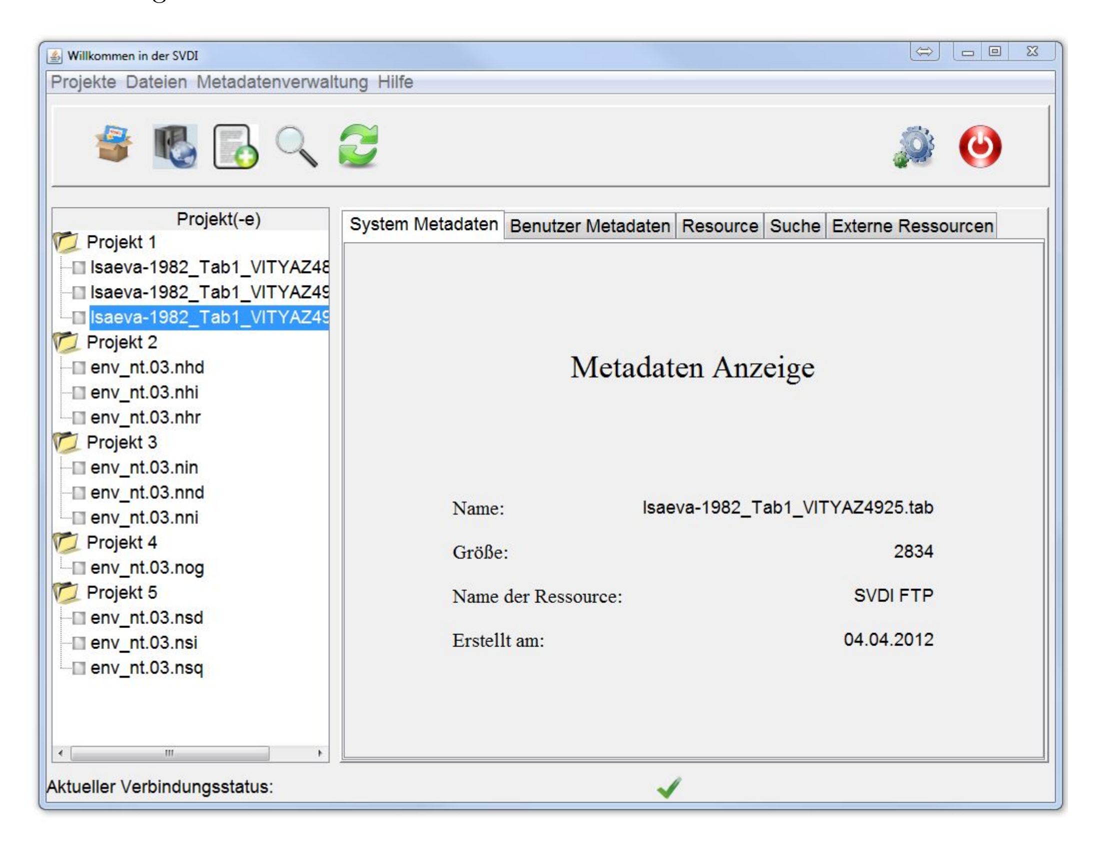


Abbildung 5.1: Systemmetadaten

Gleiches Vorgehensweise macht man, wenn man die benutzerdefinierte Metadaten einer Datei anschauen will5.2. Anstatt der Eintrag „Systemmetadaten anzeigen“, wählt man in deiser Fall „Benutzermetadaten anzeigen“. Es ist zu beachten, das in Fall, wenn die Datei keine benutzerdefinierten metadaten enthält, wird entsprechen die leere Tab

„Benutzermetadaten“ ausgegeben, anderfalls sieht man die entsprechende Metadaten in „Schlüßel-Wert“ Form. Die nicht aktuelle Benutzermetadaten können auch geändert bzw. aktualisiert werden. Dafür benutzt man die pop-Up Menü Eintrag „Benutzermetadaten andern“. Es wird eine Wizzard gestertet, wo die entsprechende Werte von Benutzer eingegeben werden können.

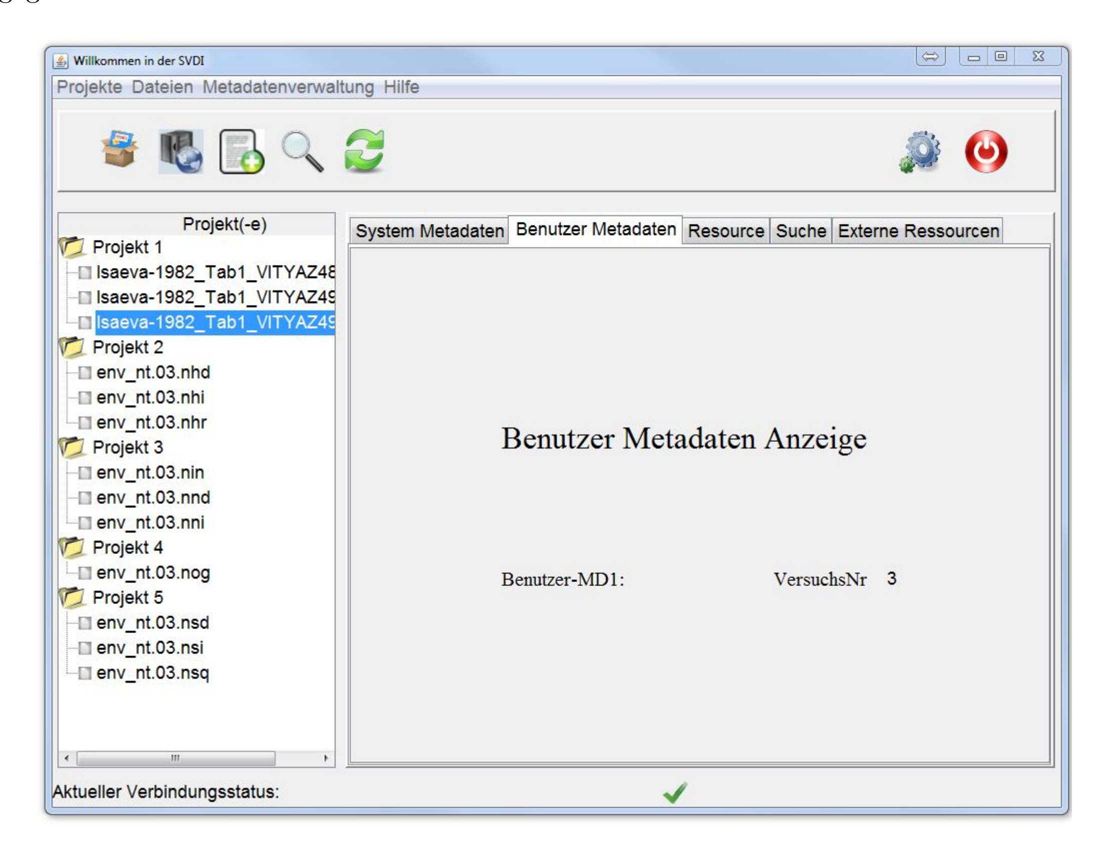


Abbildung 5.2: Vom Benutzer hinzugefügte Metadaten

Die letzte Funktionalität, der SVDI-Client zur Verfügung stellt ist die Suche. Es besteht die Möglichkeit sowohl nach die lokale Dateien, als auch an die Dateien an der eingebundenen Ressourcen zu suchen. Für die Suche steht das Tab „Suche“ zur Verfügung 5.3. Hier besteht die Möglichkeit mittels entspechenden Einträgen die Suche zu gestalten. Es ist ein Art der Filter implementiert wurde. Das heißt, je mehr Parametern man in Suchmaske eingibt, desto eingeschränkter wird er Suchergebniss sein. Es beshet die Möglichkeit, sowohl mittels benutzerdefinierten Metadaten als auch Systemmetadaten die Suche zu präzisieren.

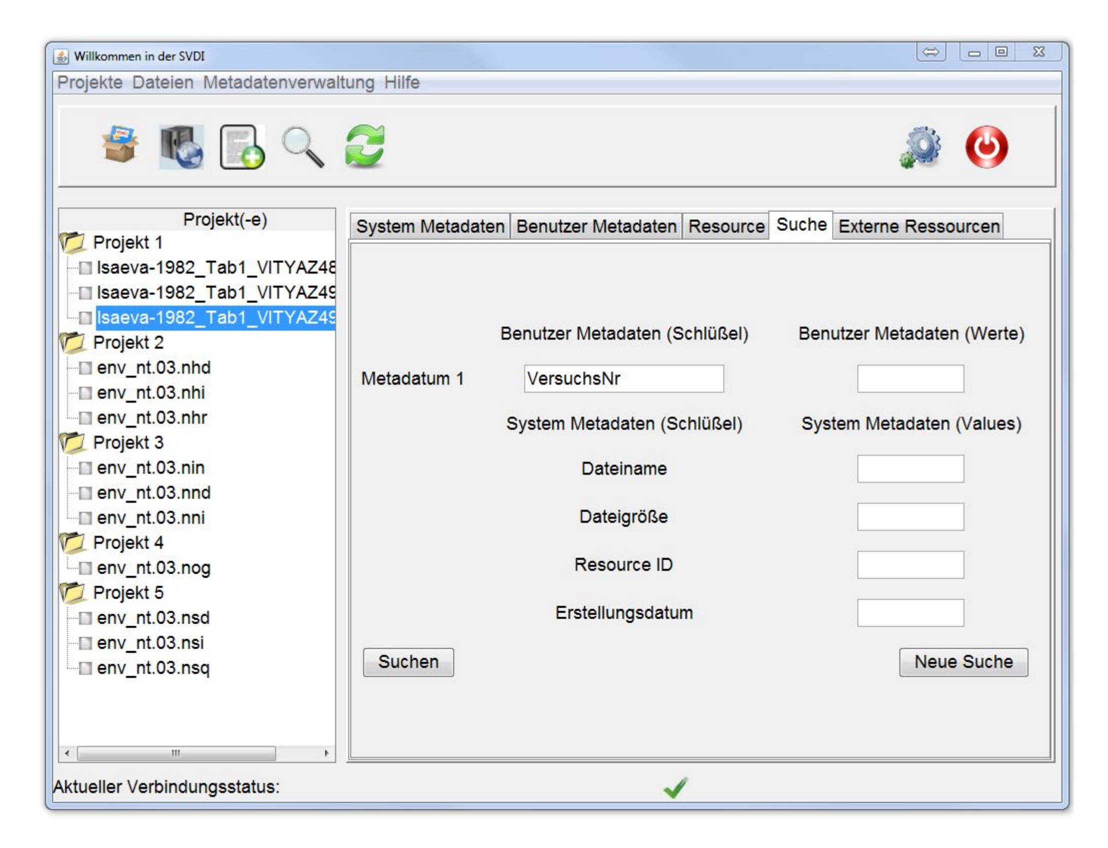


Abbildung 5.3: Suchfenster der Metadatensuche

Als Suchergebniss wird eine Tabelle ausgegeben 5.4. Hier steht die Information, wie die Datenname, Pfad zur Ressource, wo dieser Datei sich befindet, und die benutzerdefinierten Metadaten als Schlüßell-Wert Paar. Mittels Pop-Up Menü Eintrag „Ausgewahlte Datei in Projekt hinzufugen“ lässen sich hier gefundene Dateien in Projekt hinzufügen.

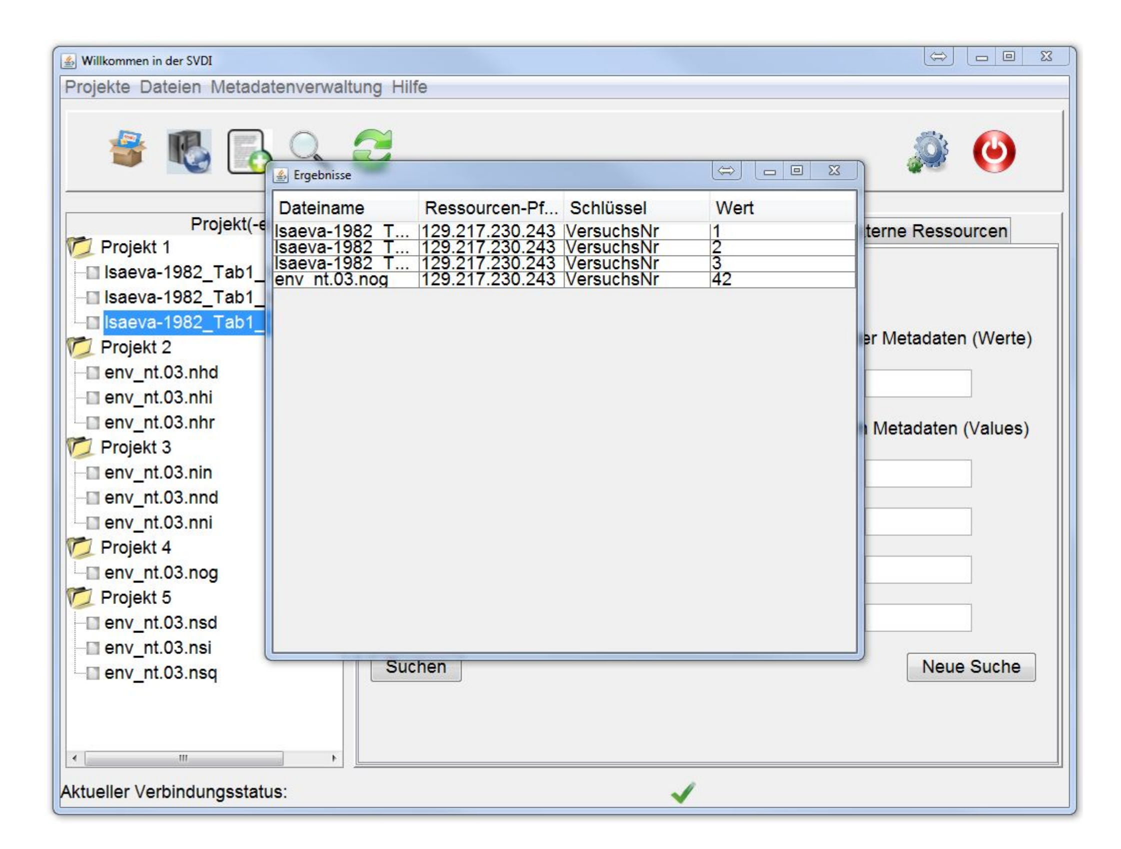


Abbildung 5.4: Suchergebnisse